



sgcWebSockets 2026.6

June 2026

CBuilder のドキュメント

Copyright © 2012-2026 eSeGeCe Software

info@esegece.com

www.esegece.com

目次

はじめに	23
概要	28
エディション	28
インストール	30
インストール	33
Install Setup	33
パッケージのインストール.....	44
Install Errors.....	51
Configure Install	55
sgcIndy パッケージのインストール.....	57
Configure ZLib	63
クイックスタート	64
概要	64
QuickStart WebSockets	66
QuickStart HTTP	68
スレッディングフロー	70
ビルド.....	72
OSXアプリケーションのビルド.....	73
Build Android Application	75
iOS アプリケーションのビルド	76
高速パフォーマンスサーバー	78
メモリマネージャー	82
OpenSSL	85
OpenSSL Windows	88
OpenSSL OSX.....	90

OpenSSL Android	92
OpenSSL iOS	93
OpenSSL 独自 CA 証明書	95
OpenSSL P12 証明書	97
OpenSSL 証明書の検証	98
OpenSSL 追加関数の読み込み	99
Indy	101

トピック 103

WebSocket イベント	103
WebSocket パラメーター接続	104
DLL内での使用	105
Windows サービス	106
Web Browser Test	107
カスタムサブプロトコル	108
認証	109
セキュア接続	111
HeartBeat	113
WatchDog	114
ログ	115
HTTP	116
ブロードキャストとチャンネル	117
バインディング	118
大容量ファイルの送信	119
圧縮	121
Flash	122
カスタムオブジェクト	123
グループ	124
IOCP	126
EPOLL	127
ALPN	128

HTTP リクエストの転送.....	129
Quality Of Service	131
キュー	133
Transactions	135
TCP 接続.....	137
SubProtocol	138
スロットル	139
Server-Sent Events	140
LoadBalancing	142
ファイル	143
プロキシ	144
フラグメント化されたメッセージ	145
コンポーネント	146
TsgcWebSocketClient.....	146
WebSocket サーバーへの接続	148
クライアントオープン接続.....	149
クライアント接続のクローズ	151
Client Keep Connection Open	152
ドロップされた切断	153
TCP サーバーへの接続	154
Connections TIME_WAIT	155
WebSocket Redirections.....	156
セキュアサーバーへの接続.....	157
証明書 OpenSSL.....	158
証明書 SChannel	159
SChannel 接続情報の取得	160
クライアントがテキストメッセージを送信する	161
クライアントによるバイナリメッセージの送信	162
クライアントのテキストおよびバイナリメッセージの送信	163
テキストメッセージを受信.....	164

バイナリメッセージの受信.....	165
クライアント認証.....	166
クライアント例外.....	169
クライアント WebSocket ハンドシェイク.....	170
クライアント登録プロトコル.....	171
クライアントプロキシ	172
TsgcTCPClient	173
TsgcTCPClient	174
TsgcWebSocketServer	175
サーバースタート.....	177
サーバーバインディング	178
Server Startup Shutdown.....	179
サーバーのアクティブ維持.....	180
サーバー SSL.....	181
Server SSL SChannel.....	183
サーバー証明書の検証	186
サーバーが接続を維持する.....	187
サーバー プレーン TCP	188
サーバーが接続を閉じる	189
クライアント接続.....	190
Server Authentication.....	191
サーバー テキストメッセージ送信.....	192
サーバーバイナリメッセージの送信.....	193
サーバーがテキストメッセージを受信.....	194
Server Receive Binary Message.....	195
サーバーがクライアントからヘッダーを読み取る	196
TsgcWebSocketHTTPServer	197
HTTP サーバーリクエスト.....	199
HTTP ディスパッチファイル.....	200
HTTP/2 サーバー	201
HTTP/2 サーバープッシュ	202

HTTP/2 Alternate Service.....	204
HTTP/2 サーバースレッド	205
レスポンスボディなしのHTTP 404エラー	207
HTTP サーバーセッション	208
HTTPサーバーストリームビデオ	210
Server SSL SChannel	211
TsgcWebSocketServer_HTTPAPI	214
HTTPAPI URL Reservation	216
HTTPAPI ファインチューン	218
HTTPAPI サーバー SSL.....	220
自己署名証明書	221
HTTPAPI HTTP/2の無効化	222
HTTPAPI Custom Headers.....	223
HTTPAPI テキストレスポンスの送信	224
HTTPAPI Send File Response	225
HTTPAPI OnDisconnect が発火しない.....	227
TsgcWebSocketClient_WinHTTP	228
TsgcWebSocketFirewall	229
ファイアウォール: ブラックリストとホワイトリスト	236
ファイアウォール: ブルートフォース保護	238
ファイアウォール: SQL インジェクションと XSS 検出	240
ファイアウォール: レート制限とフラッド防御.....	243
ファイアウォール: ボット検出	246
TsgcWSRateLimiter	249
TsgcWSCircuitBreaker.....	261
TsgcWSAPIKeyManager	272
TsgcWebSocketLoadBalancerServer.....	284
TsgcWebSocketProxyServer.....	285
TsgcIWWebSocketClient	286
TsgcWSConnection.....	288
Protocols	290

プロトコル Javascript	292
プロトコル MQTT.....	295
TsgcWSPClient_MQTT	297
クライアント MQTT 接続	298
Mosquitto MQTT サーバーへの接続	299
クライアント MQTT セッション	300
クライアント MQTT バージョン	301
MQTT パブリッシュ/サブスクライブ	302
MQTT トピック	303
MQTT Subscribe	305
MQTT Publish Message	306
MQTT メッセージの受信	307
MQTT Publish and Wait Response	308
MQTT Clear Retained Messages.....	309
プロトコル AMQP	310
TsgcWSPClient_AMQP.....	311
クライアント AMQP 接続.....	314
Client AMQP Disconnect.....	315
AMQP チャンネル	316
AMQP Exchanges	318
AMQP キュー.....	320
AMQP パブリッシュメッセージ.....	323
AMQP Consume Messages	324
AMQP メッセージの取得	326
AMQP QoS.....	327
AMQP トランザクション	328
プロトコル Kafka	330
TsgcWSPClient_Kafka.....	331
Client Kafka Connect.....	333
クライアント Kafka 切断	334
Kafka メッセージの生成	335

Kafkaメッセージの消費	336
Kafka コンシューマーグループ	337
Kafka トピック	338
Kafka オフセット	339
Protocol AMQP1	340
TsgcWSPClient_AMQP1	342
クライアント AMQP1 接続.....	345
クライアント AMQP1 切断.....	346
クライアント AMQP1 アイドルタイムアウト接続	347
クライアント AMQP1 接続状態	348
クライアント AMQP1 認証.....	349
クライアント AMQP1 Azure MessageBus	350
AMQP1 セッション	352
AMQP1 Links.....	354
AMQP1 送信者リンク	355
AMQP1 レシーバーリンク	358
AMQP1 Send Message.....	360
AMQP1 メッセージ読み取り	362
Protocol STOMP	363
TsgcWSPClient_STOMP	364
TsgcWSPClient_STOMP_RabbitMQ.....	365
TsgcWSPClient_STOMP_ActiveMQ	366
Protocol AppRTC	367
TsgcWSPServer_AppRTC.....	368
プロトコル WebRTC	369
TsgcWSPServer_WebRTC.....	370
プロトコル WebRTC Javascript.....	371
プロトコル WAMP.....	372
TsgcWSPServer_WAMP	374
TsgcWSPClient_WAMP	375
プロトコル WAMP JavaScript.....	376

サブスクライバー	379
パブリッシャー	380
シンプルRPC	381
RPC進捗結果	382
Protocol WAMP2	384
TsgcWSPClient_WAMP2	385
プロトコルデフォルト	388
TsgcWSPServer_sgc	390
TsgcWSPClient_sgc	391
TsgcIWWSPClient_sgc	392
プロトコルデフォルト JavaScript	393
プロトコルデータセット	397
TsgcWSPServer_Dataset	398
TsgcWSPClient_Dataset	399
TsgcIWWSPClient_Dataset	400
プロトコルデータセット Javascript	401
プロトコルデータセット テーブルの複製	404
プロトコルデータセット更新の通知	405
プロトコルファイル	406
TsgcWSPServer_Files	407
TsgcWSPClient_Files	408
TsgcWSMessageFile	409
サーバーへのファイルの送信方法	410
クライアントへのファイル送信方法	411
大きなファイルの送信方法	412
プロトコル プレゼンス	413
TsgcWSPServer_Presence	414
TsgcWSPPresenceMessage	415
TsgcWSPClient_Presence	416
プロトコルプレゼンス JavaScript	417
プロトコル E2EE	420

TsgcWSPServer_E2EE	422
TsgcWSPClient_E2EE	423
API	424
API Binance	426
Binance Connect WebSocket API	434
Binance WebSocket チャンネルのサブスクリプション	435
Binance マーケットデータの取得	436
Binance プライベート REST API	437
Binance スポット取引	438
Binance プライベートリクエストの時刻	440
Binance 出金	441
API Binance Futures	442
API Binance Futures 取引	448
API SocketIO	449
API Coinbase	451
Coinbase WebSocket API接続	456
Coinbase WebSocketチャンネル購読	457
Coinbase市場データ取得	458
CoinbaseプライベートREST API	459
Coinbaseプライベートリクエスト時間	460
Coinbase注文実行	461
Coinbaseサンドボックスアカウント	462
API SignalRCore	463
API SignalR	469
API Kraken	472
API Kraken WebSockets パブリック	474
API Kraken WebSockets プライベート	481
API Kraken REST パブリック	484
API Kraken REST プライベート	486
API Kraken Futures	489
API Kraken Futures WebSockets パブリック	491

API Kraken Futures WebSockets プライベート	498
API Kraken Futures REST Public	504
API Kraken Futures REST プライベート	506
API Pusher	511
API Bitmex	520
Bitmex WebSocket API への接続	524
Bitmex Subscribe WebSocket Channel	525
Bitmex 注文の発注方法	526
API Bitfinex	528
API Kucoin	532
Kucoin WebSocket API への接続	539
Kucoin Subscribe WebSocket Channel	540
Kucoin 市場データの取得	541
Kucoin Private REST API	542
Kucoin Trade Spot	543
Kucoin プライベートリクエストの時間	545
API Kucoin 先物	546
Kucoin Futures WebSocket API への接続	552
Kucoin Futures Subscribe WebSocket Channel	553
Kucoin Futures 市場データの取得	554
Kucoin Futures プライベート REST API	555
Kucoin Futures 取引	556
Kucoin Futures Private Requests Time	559
API 3Commas	560
API OKX	566
API XTB	572
API Bybit	575
API Forex	580
Forex Connect REST API	582
Forex 取引注文	583
外国為替アカウントクエリ	585

外国為替マーケットデータ	586
Forex Connect WebSocket API	587
Forex ストリームのサブスクライブ	588
API Blockchain	590
API Cex.....	592
API Cex Plus	599
API Discord.....	604
API OpenAI	607
API MEXC	609
API MEXC Futures.....	613
API Bitget.....	617
API GateIO	620
API Deribit	622
API Crypto.com.....	625
API HTX	628
API Whatsapp	630
WhatsApp アプリの作成	634
WhatsApp 電話番号 ID	636
WhatsApp トークン	637
WhatsApp Webhook	638
WhatsApp セキュリティ	639
WhatsApp メッセージの送信.....	640
WhatsApp Send Interactive Messages.....	643
WhatsApp テンプレートメッセージの送信	648
WhatsApp メッセージの受信とステータス通知.....	650
WhatsApp ファイル送信	652
WhatsApp Download Media	654
API Telegram.....	655
インラインボタン付きの Telegram メッセージを送信する	664
ボタン付きの Telegram メッセージを送信する	665
Telegram 太字メッセージの送信.....	666

ボットとして Telegram チャットが見つかりません.....	667
Telegram スポンサーメッセージ.....	668
Send Telegram Invoice Message.....	669
Telegramでスーパーグループメンバーを取得する.....	670
Telegram プロキシを追加します.....	671
Telegram ユーザーを登録する.....	672
RCON	673
CryptoHopper.....	674
RTCMultiConnection	679
WebPush	681
TsgcWSAPIServer_WebPush	682
TsgcWebPush_Client.....	683
Extensions.....	684
PerMessage-Deflate.....	685
Deflate-Frame.....	686
MCP.....	687
TsgcWSAPIServer_MCP.....	688
MCP Server Sessions.....	690
MCP Server Tools	691
MCP サーバープロンプト	694
MCP サーバーリソース.....	697
MCP Server Roots.....	700
MCP サーバーサンプリング	701
MCP サーバーエリシテーション.....	703
TsgcWSAPIClient_MCP	705
MCP クライアントツール.....	706
MCP クライアントプロンプト	708
MCP クライアントリソース	710
MCP クライアントルーツ	712
MCP クライアントサンプリング.....	713
MCPクライアント エリシテーション	714

OpenAI.....	715
TsgcHTTP_API_OpenAI.....	722
OpenAI Completion	723
OpenAI チャット.....	724
OpenAI Edit.....	725
OpenAI Audio.....	726
OpenAI モデレーション	727
OpenAI RealTime.....	728
OpenAI レスpons	729
OpenAI Speech	730
OpenAI ファインチューニング	731
OpenAI Batch.....	732
OpenAI アップロード	733
OpenAI アプリケーション.....	734
TsgcAIOpenAIAssistant.....	735
TsgcAIOpenAIAssistant ファイル検索	737
TsgcAIOpenAIAssistant ストリーミング.....	739
TsgcAIOpenAIAssistant 関数呼び出し	741
OpenAI Audio.....	743
TsgcAudioRecorderMCI	744
TsgcAudioPlayerMCI	745
TsgcTextToSpeechSystem.....	746
TsgcTextToSpeechGoogle	747
TsgcTextToSpeechAmazon	748
TsgcAIChat.....	749
TsgcAIOpenAIChatBot	750
TsgcAIOpenAITranslator.....	751
TsgcAIOpenAIEmbeddings.....	752
TsgcAIDatabaseVectorFile.....	753
TsgcAIDatabaseVectorPinecone.....	754
埋め込みベクターの作成	755

埋め込み ChatBot.....	756
Pinecone.....	757
Anthropic.....	760
TsgcHTTP_API_Anthropic.....	764
Anthropic メッセージ	765
Anthropic ビジョン	767
Anthropic Tool Use	769
Anthropic モデル	771
Anthropic バッチ	772
Anthropic 拡張思考	773
Anthropic Documents.....	775
Anthropic プロンプトキャッシング	777
Anthropic Citations	779
Anthropic ウェブ検索	780
Anthropic Structured Outputs.....	782
Anthropic ファイル.....	784
Anthropic MCP コネクタ	786
Gemini	788
TsgcHTTP_API_Gemini	790
DeepSeek	791
TsgcHTTP_API_DeepSeek	792
DeepSeek メッセージ.....	793
DeepSeek Vision.....	794
DeepSeek モデル	795
Gemini	796
Gemini メッセージ	798
Gemini Vision.....	800
Gemini モデル	801
Gemini 構造化出力	802
Gemini トークンカウント	803
Gemini 埋め込み	804

Gemini ツール使用	805
Ollama	807
TsgcHTTP_API_Ollama	809
Ollama メッセージ.....	810
Ollama Models.....	811
Ollama 埋め込み	812
Grok	813
TsgcHTTP_API_Grok	814
Grok メッセージ.....	815
Grok Vision.....	816
Grok モデル	817
Mistral AI	818
TsgcHTTP_API_Mistral.....	820
Mistral メッセージ	821
Mistral ビジョン	822
Mistral モデル.....	823
Mistral 埋め込み.....	824
IoT	825
IoT Amazon MQTT クライアント	826
IoT Azure MQTT クライアント	834
HTTP.....	839
HTTP2	840
TsgcHTTP2Client.....	841
HTTP/2 リクエストメソッド	843
HTTP/2 サーバープッシュ	844
HTTP/2 ファイルのダウンロード	845
HTTP/2 部分レスポンス.....	846
HTTP/2 ヘッダー	847
クライアント接続のクローズ.....	848
クライアント接続維持	849
HTTP/2 切断理由	850

クライアント保留リクエスト	851
クライアント認証	852
HTTP/2 と OAuth2	853
TsgcHTTP2ConnectionClient	854
TsgcHTTP2RequestProperty	855
TsgcHTTP2ResponseProperty	856
Apple Push Notifications	857
リモート通知APNsの生成	858
APNs への通知リクエストの送信	859
APNs へのトークンベース接続	860
APNs への証明書ベースの接続	861
HTTP1	863
OAuth2	867
TsgcHTTP_OAuth2_Client	868
Web アプリケーション用 OAuth2 クライアント	872
デスクトップアプリケーション向け OAuth2 クライアント	873
TsgcHTTP_OAuth2_Client_Google	874
TsgcHTTP_OAuth2_Client_Microsoft	875
Authorization Code Grant (RFC 6749)	876
PKCE を使用した認可コード (RFC 7636)	878
Client Credentials Grant (RFC 6749)	880
Resource Owner Password Credentials グラント (RFC 6749)	882
デバイス認証グラント (RFC 8628)	884
DPoP - Demonstrating Proof of Possession (RFC 9449)	886
TsgcHTTP_OAuth2_Server	890
OAuth2 サーバーの例	893
OAuth2 サインインHTMLのカスタマイズ	897
OAuth2 サーバーエンドポイント	898
OAuth2 アプリの登録	899
OAuth2 アクセストークンの回復	900
OAuth2 サーバー認証	901

OAuth2 なし 認証URL.....	902
認可コードグラント	903
クライアント資格情報グラント	905
リソースオーナーパスワード認証情報グラント	907
Refresh Token グラント	909
デバイス認証グラント (RFC 8628)	911
Token Revocation (RFC 7009)	913
Token Introspection (RFC 7662)	915
DPoP 検証 (RFC 9449)	917
TsgcHTTP_OAuth2_Server_Provider	919
OAuth2 プロバイダー Azure AD	921
OAuth2 プロバイダーのプライベートエンドポイント	922
OAuth2 Provider Authentication	923
OAuth2 プロバイダーリクエスト	925
JWT	926
TsgcHTTP_JWT_Client	928
TsgcHTTP_JWT_Server.....	931
WebAuthn	932
TsgcWSAPIServer_WebAuthn	933
WebAuthn 登録.....	937
WebAuthn Registration Request	940
WebAuthn登録レスポンス	941
WebAuthn 登録結果	943
WebAuthn 認証.....	944
WebAuthn 認証リクエスト	946
WebAuthn 認証レスポンス	947
WebAuthn認証結果	948
WebAuthn MDS	949
WebAuthn 認証.....	950
WebAuthn Authorization HTTP	951
WebAuthn 認証 WebSocket	952

WebAuthn Javascript クライアント	953
Amazon SQS	957
Google OAuth2 キー	962
Google サービスアカウント	968
Google Cloud Pub/Sub	972
Google Calendar	981
Google Calendar Sync Calendars	987
Google Calendar 同期イベント	988
Google Calendar RefreshToken.....	989
Google Calendar サービスアカウント	990
Google Cloud FCM.....	991
TsgcWebView2	994
WebView2 Navigation.....	997
WebView2 JavaScript.....	999
WebView2 クッキー.....	1001
WebView2 ダウンロード	1003
WebView2 設定	1005
WebView2 高度な機能.....	1007
TsgcUDPCliant	1011
TsgcUDPServer	1012
STUN	1013
TsgcSTUNClient	1014
STUN クライアント UDP 再送信.....	1016
STUN クライアント長期資格情報	1017
STUN クライアント属性	1018
TsgcSTUNServer	1019
STUNサーバー長期資格情報.....	1020
STUN サーバー代替サーバー.....	1021
TURN.....	1022
TsgcTURNClient	1023
TURN クライアントの IP アドレスのアロケーション	1025

TURN クライアント パーミッションの作成	1026
TURN クライアント 送信インジケーション	1027
TURN Client Channels.....	1028
TsgcTURNServer	1029
TURNサーバー長期認証情報	1031
TURN サーバー Allocation.....	1032
ICE	1033
TsgcICEClient.....	1034
ICE候補の収集	1035
ICE ペア候補	1036
TsgcRTCPeerConnection	1037
RTCPeerConnection WebSocket サーバー	1039
RTCPeerConnection WebSocket クライアント	1040
RTCPeerConnection STUN TURN	1041
RTCPeerConnection シグナリング	1042
RTCPeerConnection ICE.....	1043
RTCPeerConnection DTLS	1044
RTCPeerConnection データ	1045
Datasnap	1046
TsgcWSHTTPWebBrokerBridgeServer	1047
TsgcWSHTTP2WebBrokerBridgeServer	1049
TsgcWSServer_HTTPAPI_WebBrokerBridge.....	1050

OpenAPI 1051

OpenAPI	1051
OpenAPI パーサー Pascal.....	1053
OpenAPI 追加プロパティ	1058
OpenAPI コマンドライン	1060
OpenAPI ライブラリ	1062
OpenAPI API.....	1064
OpenAPI クライアント	1065

OpenAPI サーバー	1068
OpenAPI サーバースペックファースト	1070
OpenAPI サーバー コードファースト	1071
OpenAPI Amazon AWS	1072
OpenAPI Amazon AWS 認証情報	1078
OpenAPI Amazon AWS S3	1080
OpenAPI Google Cloud	1081
OpenAPI Google Cloud OAuth2	1086
OpenAPI Google Cloud サービスアカウント	1089
OpenAPI Google Cloud PubSub	1094
OpenAPI Google Cloud Calendar	1095
OpenAPI Google Drive	1096
OpenAPI Microsoft	1098
OpenAPI Microsoft テナント	1103
OpenAPI Microsoft アプリケーションの登録	1104
OpenAPI Microsoft OAuth2 コード	1107
OpenAPI Microsoft OAuth2 資格情報	1109
OpenAPI Microsoft Graph	1111
API	1112
AbstractApi Geolocation	1113

デモ 1114

サーバーチャット	1114
クライアントチャット	1116
クライアント	1117
クライアント MQTT	1118
クライアント SocketIO	1120
サーバーモニター	1121
サーバースナップショット	1124
Client Snapshots	1125
ファイルのアップロード	1126

Server Authentication.....	1128
KendoUI_Grid.....	1129
ServerSentEvents	1131
サーバー WebRTC	1132
サーバー AppRTC	1133
Telegram Client	1135
サードパーティ	1137
Coturn.....	1137
ライセンス	1140
ライセンス	1140
インデックス	1142
PDF-Back-Cover	1151

はじめに

sgcWebSocketsは、リアルタイム通信、メッセージング、AI統合のためのプロフェッショナルグレードのコンポーネントライブラリです。WebSocket、HTTP/2、MQTT、AMQP、WebRTC、Server-Sent Events、および30を超えるサードパーティAPIコネクタの本番環境対応の実装を、一貫したイベント駆動型のコンポーネントアーキテクチャを通じてすべて利用可能にします。トレーディングプラットフォーム、IoTゲートウェイ、AI搭載アプリケーション、またはエンタープライズメッセージングシステムのいずれを構築する場合でも、sgcWebSocketsは必要なネットワーキング基盤を提供します。

このライブラリは **Delphi 7 から Delphi 13**、**C++Builder**、**Lazarus/FreePascal**、および **.NET** (Framework 2.0+、.NET Core 1.0+、.NET 5-9、.NET Standard) をサポートします。アプリケーションは VCL と FireMonkey を使用して単一のコードベースから Windows、macOS、Linux、iOS、Android をターゲットにできます。

Delphi / C++Builder / FreePascal Features

コア WebSocket

- **RFC 6455** を実装した完全機能のマルチスレッド **WebSocket** サーバー。
- テキストおよびバイナリメッセージのサポート。
- PerMessage_Deflate拡張機能 (**RFC 7692**) を使用した**メッセージ圧縮**です。
- 自動再接続と接続監視のための組み込み **WatchDog** および **HeartBeat**。
- **ロードバランシング**サーバーで複数のバックエンドに接続を分散します。
- **プロキシ**サーバー コンポーネント。ウェブブラウザが任意の TCP サーバーに接続できるようにします。
- **HTTP プロキシ**および**SOCKS プロキシ**サーバー経由のクライアント接続。
- イベント: OnConnect、OnDisconnect、OnMessage、OnError、OnHandshake。

高性能サーバー

- **Microsoft HTTP Server API** (HTTP.SYS) と高パフォーマンス Windows サーバー向けの **IOCP**。
- **IOCP** (Windows)、**EPOLL** (Linux)、またはデフォルトの 1 接続 1 スレッド モデルを使用した Indy サーバー。
- 同一ポートを通じた **WebSocket** と **HTTP/2** 接続。
- **WebBroker** サーバーは単一ポートで **DataSnap**、**HTTP/2**、および **WebSocket** をサポートします。
- **WinHTTP API** ベースのクライアント WebSocket。

セキュリティと認証

- サーバーおよびクライアントコンポーネント用の **SSL/TLS**。 **OpenSSL 1.1.1** および **3.0** をサポート。クライアントは Windows で **SChannel** をサポートします。
- **OAuth 2.0** および **JWT** (JSON Web Token) のクライアントおよびサーバーコンポーネント。
- **WebAuthn Server Protocol** パスキーベースの認証用。
- **E2EE** (エンドツーエンド暗号化) サブプロトコル。

AI と機械学習

- **OpenAI** および **Claude** の **MCP** サーバーおよび **MCP** クライアントコンポーネントとの統合。
- **ChatBot** および **Translator** アプリケーションコンポーネント (音声コマンドサポート付き)。
- **エンベディング**とベクターデータベースのサポート (ファイルベースおよび Pinecone)。
- **テキスト読み上げ (Text-To-Speech)** はシステムデフォルト、Google Cloud、Amazon AWSを介して利用できます。

HTTP/2 プロトコル

- 完全な **HTTP/2**クライアントおよびサーバー実装。
- HTTP上の**Server-Sent Events** (プッシュ通知)。

メッセージングプロトコル

- WebSocket およびプレーン TCP 上の **MQTT 3.1.1** および **5.0**。
- **AMQP 0.9.1** および **1.0** (RabbitMQ ブローカーサポート付き)。
- **STOMP** (WebSocket および TCP、ActiveMQ、RabbitMQ ブローカー対応)。
- **WAMP 1.0** および **2.0** のクライアントおよびサーバープロトコル。
- **JSON-RPC 2.0**サブプロトコル。

P2P とリアルタイム通信

- **WebRTC** サーバープロトコル。
- **STUN**および**TURN**のクライアントおよびサーバーコンポーネント。
- **ICE NAT** トラバーサル用のクライアント。
- **UDP** クライアントとサーバー。

API インテグレーション

- **30 以上の組み込み API コネクター**: Binance、Coinbase、Kraken、Bitfinex、Discord、WhatsApp、Telegram など。
- **SignalR** および **SignalR Core**、**Socket.IO**、および **Pusher** クライアント。
- クラウドサービス: **Amazon SQS**、**Google Cloud Pub/Sub**、**Google Calendar**。

クロスプラットフォーム

- **VCL** および **FireMonkey** (Windows、macOS、iOS、Android、Linux)。

- **C++Builder** サポート。
- **Lazarus / FreePascal** サポート。
- ブラウザクライアント用の組み込み **JavaScript ライブラリ**。

sgcWebSockets ライブラリに含まれるコンポーネントを以下に示します。

1 sgcWebSockets

- **TsgcWebSocketClient**: Indy ライブラリをベースにした WebSocket クライアント。
- **TsgcWebSocketServer**: Indy ライブラリをベースにした WebSocket サーバー
- **TsgcWebSocketHTTPServer**: Indy ライブラリをベースにした WebSocket + HTTP サーバー。
- **TsgcWebSocketServer_HTTPAPI**: HTTP.SYS Microsoft HTTP API をベースにした高性能 WebSocket + HTTP サーバー。
- **TsgcWebSocketClient_WinHTTP**: WinHTTP ライブラリをベースにした WebSocket クライアント。
- **TsgcWebSocketFirewall**: IP フィルタリング、ブルートフォース保護、レートリミット、脅威検出を備えた WebSocket サーバーファイアウォール。
- **TsgcWebSocketLoadBalancerServer**: WebSocket および HTTP プロトコル向けのロードバランサーサーバー。
- **TsgcWebSocketProxyServer**: ウェブブラウザが TCP サーバーに接続できるようにする WebSocket プロキシサーバー。

2 sgcWebSocket API

- **TsgcWSAPI_Binance**: Binance スポットクライアント。WebSocket + REST API をサポート。
- **TsgcWSAPI_Binance_Futures**: Binance フューチャーズクライアント。WebSocket + REST API をサポート。
- **TsgcWSAPI_SocketIO**: Socket.IO クライアント。
- **TsgcWSAPI_Coinbase**: Coinbase Pro クライアント。WebSocket + REST API をサポート。
- **TsgcWSAPI_Bitmex**: Bitmex クライアント。WebSocket + REST API をサポート。
- **TsgcWSAPI_SignalR**: SignalR WebSocket クライアント。
- **TsgcWSAPI_SignalRCore**: SignalRCore WebSocket クライアント。
- **TsgcWSAPI_Pusher**: Pusher WebSocket クライアント。
- **TsgcWSAPI_Kraken**: Kraken クライアント API。WebSocket および REST API をサポート。
- **TsgcWSAPI_Kraken_Futures**: Kraken フューチャーズクライアント API。WebSocket および REST API をサポート。
- **TsgcWSAPI_Bitstamp**: Bitstamp WebSocket クライアント。
- **TsgcWSAPI_Cex**: Cex WebSocket クライアント。
- **TsgcWSAPI_Huobi**: Huobi WebSocket クライアント。
- **TsgcWSAPI_ThreeCommas**: ThreeCommas クライアント API。
- **TsgcWSAPI_Bitfinex**: Bitfinex WebSocket API。
- **TsgcWSAPI_Discord**: Discord WebSocket クライアント。
- **TsgcWSAPI_BlockChain**: BlockChain WebSocket クライアント。
- **TsgcWSAPI_Bybit**: Bybit WebSocket クライアント。
- **TsgcWSAPI_OKX**: OKX WebSocket クライアント。
- **TsgcWSAPI_Kucoin**: KuCoin スポット WebSocket クライアント。

- [TsgcWSAPI_Kucoin_Futures](#): KuCoin フューチャーズ WebSocket クライアント。
- [TsgcWSAPI_Deribit](#): Deribit WebSocket クライアント。
- [TsgcWSAPI_CryptoCom](#): Crypto.com WebSocket クライアント。
- [TsgcWSAPI_HTX](#): HTX WebSocket クライアント。
- [TsgcWSAPI_MEXC](#): MEXC WebSocket クライアント。
- [TsgcWSAPI_Bitget](#): Bitget WebSocket クライアント。
- [TsgcWSAPI_GateIO](#): Gate.io WebSocket クライアント。
- [TsgcWSAPI_XTB](#): XTB WebSocket クライアント。
- [TsgcWSAPI_CexPlus](#): CexPlus WebSocket クライアント。
- [TsgcWSAPI_OpenAI](#): OpenAI リアルタイム WebSocket API クライアント。

3 sgcWebSocket Libs

- [TsgcTDLib_Telegram](#): Telegram APIクライアント。
- [TsgcWhatsApp_Client](#): WhatsApp Business Cloudクライアント。
- [TsgcHTTP_Cryptohopper](#): CryptohopperクライアントAPI。
- [TsgcLib_RCON](#): RCONクライアント。

4 sgcWebSocket プロトコル

- [TsgcWSPClient_MQTT](#): MQTT (3.1.1 および 5.0) クライアント。WebSocket およびプレーン TCP 接続をサポートします。
- [TsgcWSPClient_AMQP1](#): AMQP 1.0.0 クライアント。RabbitMQ ブローカーをサポートします。
- [TsgcWSPClient_AMQP](#): AMQP 0.9.1 クライアント。RabbitMQ ブローカーをサポートします。
- [TsgcWSPClient_STOMP](#): STOMP クライアント。WebSocket およびプレーン TCP 接続をサポートします。
 - [TsgcWSPClient_STOMP_ActiveMQ](#): ActiveMQ ブローカー用 STOMP クライアント。
 - [TsgcWSPClient_STOMP_RabbitMQ](#): RabbitMQ ブローカー用 STOMP クライアント。
- [TsgcWSPClient_WAMP](#): WAMP 1.0 クライアントプロトコル。
- [TsgcWSPServer_WAMP](#): WAMP 1.0 サーバプロトコル。
- [TsgcWSPClient_WAMP2](#): WAMP 2.0 クライアントプロトコル。
- [TsgcWSPServer_AppRTC](#): AppRTC Google プロジェクトに基づいた WebRTC サーバー。
- [TsgcWSPServer_WebRTC](#): WebRTC サーバプロトコル。
- [TsgcWSPClient_sgc](#): JSON RPC に基づく WebSocket クライアント SGC プロトコル。
- [TsgcWSPServer_sgc](#): JSON RPC に基づく WebSocket サーバ SGC プロトコル。
- [TsgcWSPClient_Files](#): WebSocket ファイル転送クライアントプロトコル。
- [TsgcWSPServer_Files](#): WebSocket ファイル転送サーバプロトコル。
- [TsgcWSPClient_Dataset](#): WebSocket クライアントデータセット同期プロトコル。
- [TsgcWSPServer_Dataset](#): WebSocket サーバデータセット同期プロトコル。
- [TsgcWSPClient_Presence](#): WebSocket クライアントプレゼンスプロトコル。
- [TsgcWSPServer_Presence](#): WebSocket サーバプレゼンスプロトコル。
- [TsgcWSPClient_E2EE](#): エンドツーエンド暗号化クライアントプロトコル。
- [TsgcWSPServer_E2EE](#): エンドツーエンド暗号化サーバプロトコル。

5 sgcWebSockets HTTP

- **TsgcHTTP1Client**: Indy TIdHTTP に基づく HTTP 1.0 クライアント。
- **TsgcHTTP2Client**: HTTP 2.0 クライアント。
- **TsgcHTTP_JWT_Client**: JWT (JSON WEB TOKEN) クライアント。
- **TsgcHTTP_JWT_Server**: JWT (JSON WEB TOKEN) サーバー。
- **TsgcHTTP_OAuth2_Client**: OAuth 2.0 クライアント。
- **TsgcHTTP_OAuth2_Server**: OAuth 2.0 サーバー。
- **TsgcHTTPAWS_SQS_Client**: Amazon AWS SQS クライアント。
- **TsgcHTTPGoogleCloud_PubSub_Client**: Google Cloud Pub/Sub クライアント。
- **TsgcHTTPGoogleCloud_Calendar_Client**: Google カレンダークライアント。
- **TsgcHTTP_OAuth2_Client_Google**: Google 用 OAuth 2.0 クライアント。
- **TsgcHTTP_OAuth2_Client_Microsoft**: Microsoft 用 OAuth 2.0 クライアント。
- **TsgcHTTP_OAuth2_Server_Provider**: OAuth 2.0 サーバープロバイダー。
- **TsgcWSAPIServer_WebAuthn**: パスキーベース認証のための WebAuthn サーバー。
- **TsgcHTTPGoogleCloud_FCM_Client**: Google Cloud FCM (Firebase Cloud Messaging) クライアント。

6 sgcWebSockets IoT

- **TsgcloAmazon_MQTT_Client**: Amazon MQTT IoT Coreクライアント。
- **TsgcloAzure_MQTT_Client**: Azure IoT MQTTクライアント。

7 sgcWebSockets P2P

- **TsgcUDPCClient**: UDPクライアント。
- **TsgcUDPServer**: UDPサーバー。
- **TsgcSTUNClient**: STUNクライアント。
- **TsgcSTUNServer**: STUNサーバー。
- **TsgcTURNClient**: STUN / TURNクライアント。
- **TsgcTURNServer**: STUN / TURNサーバー。
- **TsgcICEClient**: ICEクライアント。
- **TsgcRTCPeerConnection**: WebRTC ピア接続クライアント。

8 sgcWebSockets DataSnap

- **TsgcWSHTTPWebBrokerBridgeServer** : HTTP + WebSockets サポート付き DataSnap サーバー代替。
- **TsgcWSHTTP2WebBrokerBridgeServer** : HTTP + HTTP/2 + WebSockets サポート付き DataSnap サーバー代替。
- **TsgcWSServer_HTTPAPI_WebBrokerBridge** : HTTP.SYS Microsoft サーバーに基づく DataSnap サーバー代替。

9 sgcWebSockets AI

- **TsgcAIOpenAIChatBot**: 音声コマンドでChatBotを構築します。
- **TsgcAIOpenAITranslator**: リアルタイム翻訳。
- **TsgcAudioRecorderMCI**: MCIを使用して音声を録音します。
- **TsgcAudioPlayerMCI**: MCIを使用して音声を再生します。
- **TsgcTextToSpeechSystem**: オペレーティングシステムのデフォルトを使用したText-To-Speech。

- **TsgcTextToSpeechGoogle:** Google Cloudを使用したText-To-Speech。
- **TsgcTextToSpeechAmazon:** Amazon AWSを使用したText-To-Speech。
- **TsgcAIOpenAIEmbeddings:** カスタムデータを使用してAIアプリケーションを構築できます。
- **TsgcAIDatabaseVectorFile:** ベクトルをプレーンテキストファイルに保存します。
- **TsgcAIDatabaseVectorPinecone:** Pineconeベクトルデータベースをサポートします。
- **TsgcAIOpenAIAssistant:** OpenAI Assistants APIクライアント。
- **TsgcWSAPIServer_MCP:** MCP (Model Context Protocol) Server。
- **TsgcWSAPIClient_MCP:** MCP (Model Context Protocol) Client。

Versions Support

Delphi サポート IDE

- Delphi 7 (* Indy 10 にアップグレードした場合にのみサポートされます。Intrawebはサポートされません)
- 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 対応 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 でサポートされている IDE

- Lazarus

試用版

無料版に含まれるコンパイル済み *.dcu ファイルは、デフォルトの Indy と IntraWeb バージョンを使用しています。これらのパッケージをアップグレードした場合、おそらく動作しないため、完全なソースコード版を購入する必要があります。

Indy パッケージ

一部のコンポーネントは TCP/IP ライブラリとして Indy を使用しています (TsgcWebsocketClient や TsgcWebsocketServer など)。これは、sgcWebSockets パッケージをインストールするために Indy が必要であることを意味します。デフォルトでは sgcWebSockets は RAD Studio に組み込まれた Indy ライブラリを使用しますが、OpenSSL API 1.1、OpenSSL 3.0、ALPN プロトコルのサポートなど、より多くの機能を持つカスタム Indy バージョンを提供しています。

インストール

Delphi / CBuilder / Lazarus

1. 付属ファイルをディレクトリ `{$DIR}` に解凍します。

2. Delphi\CBuilder から:

ファイルが解凍された `{$DIR}` ディレクトリを、ツール、環境オプション、ディレクトリの下 の Delphi/CBuilder ライブラリパスに追加してください。

すべての Delphi/CBuilder バージョン

ディレクトリ `{$DIR}\source` をライブラリパスに追加します

特定の Delphi バージョン向け

Delphi 7 : ライブラリパスにディレクトリ `{$DIR}\libD7` を追加します
 Delphi 2007 : ライブラリパスにディレクトリ `{$DIR}\libD2007` を追加します。
 Delphi 2009 : ライブラリパスにディレクトリ `{$DIR}\libD2009` を追加します
 Delphi 2010 : ライブラリパスにディレクトリ `{$DIR}\libD2010` を追加します
 Delphi XE : ディレクトリ `{$DIR}\libDXE` をライブラリパスに追加してください
 Delphi XE2 : ライブラリパスにディレクトリ `{$DIR}\libDXE2$(Platform)` を追加してください
 Delphi XE3 : ライブラリパスにディレクトリ `{$DIR}\libDXE3$(Platform)` を追加
 Delphi XE4 : ライブラリパスにディレクトリ `{$DIR}\libDXE4$(Platform)` を追加します。
 Delphi XE5 : `{$DIR}\libDXE5$(Platform)` ディレクトリをライブラリパスに追加します
 Delphi XE6 : `{$DIR}\libDXE6$(Platform)` ディレクトリをライブラリパスに追加します
 Delphi XE7 : ディレクトリ `{$DIR}\libDXE7$(Platform)` をライブラリパスに追加します
 Delphi XE8 : ディレクトリ `{$DIR}\libDXE8$(Platform)` をライブラリパスに追加します
 Delphi 10 : ライブラリパスにディレクトリ `{$DIR}\libD10$(Platform)` を追加します
 Delphi 10.1 : ディレクトリ `{$DIR}\libD10_1$(Platform)` をライブラリパスに追加します
 Delphi 10.2 : ディレクトリ `{$DIR}\libD10_2$(Platform)` をライブラリパスに追加してください
 Delphi 10.3 : ディレクトリ `{$DIR}\libD10_3$(Platform)` をライブラリパスに追加します
 Delphi 10.4 : ライブラリパスにディレクトリ `{$DIR}\libD10_4$(Platform)` を追加する
 Delphi 11 : ディレクトリ `{$DIR}\libD11$(Platform)` をライブラリパスに追加します
 Delphi 12 : ライブラリパスにディレクトリ `{$DIR}\libD12$(Platform)` を追加します。
 Delphi 13 : ライブラリパスにディレクトリ `{$DIR}\libD13$(Platform)` を追加してください

特定の CBuilder バージョン向け

C++ Builder 2010 : ライブラリパスにディレクトリ `{$DIR}\libD2010` を追加します。
 C++ Builder XE : ライブラリパスにディレクトリ `{$DIR}\libDXE` を追加します
 C++ Builder XE2 : ライブラリパスにディレクトリ `{$DIR}\libDXE2$(Platform)` を追加してください
 C++ Builder XE3 : ライブラリパスにディレクトリ `{$DIR}\libDXE3$(Platform)` を追加します。
 C++ Builder XE4 : ライブラリパスにディレクトリ `{$DIR}\libDXE4$(Platform)` を追加してください
 C++ Builder XE5 : ライブラリパスにディレクトリ `{$DIR}\libDXE5$(Platform)` を追加
 C++ Builder XE6 : ライブラリパスにディレクトリ `{$DIR}\libDXE6$(Platform)` を追加してください
 C++ Builder XE7 : ライブラリパスにディレクトリ `{$DIR}\libDXE7$(Platform)` を追加します。
 C++ Builder XE8 : ライブラリパスにディレクトリ `{$DIR}\libDXE8$(Platform)` を追加します
 C++ Builder 10 : ライブラリパスにディレクトリ `{$DIR}\libD10$(Platform)` を追加します。
 C++ Builder 10.1 : ライブラリパスにディレクトリ `{$DIR}\libD10_1$(Platform)` を追加してください
 C++ Builder 10.2 : ディレクトリ `{$DIR}\libD10_2$(Platform)` をライブラリパスに追加します
 C++ Builder 10.3 : ライブラリパスにディレクトリ `{$DIR}\libD10_3$(Platform)` を追加してください
 C++ Builder 10.4 : ライブラリパスにディレクトリ `{$DIR}\libD10_4$(Platform)` を追加してください。
 C++ Builder 11 : ライブラリパスにディレクトリ `{$DIR}\libD11$(Platform)` を追加してください。

C++ Builder 12 : ライブラリパスにディレクトリ `{$DIR}\libD12\$(Platform)` を追加してください。
 C++ Builder 13 : ディレクトリ `{$DIR}\libD13\$(Platform)` をライブラリパスに追加します

すべてのCBuilderバージョンについて、ライブラリパスに`dcp\$(Platform)`を追加してください（.bpi
 ファイルを含みます）

3. Delphi から

選択してください

ファイルを開き、正しい `Packages\sgcWebSockets.groupproj` を参照します（最初に
`sgcWebSocketsX.dpk` をコンパイルし、次に `dclsgcWebSocketsX.dpk` をインストールします）。

Delphi 用パッケージファイル

`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. CBuilder から

選択してください

ファイルを開き、正しい `Packages\sgcWebSockets.groupproj` を参照します（最初に
`sgcWebSocketsX.dpk` をコンパイルし、次に `dclsgcWebSocketsX.dpk` をインストールします）。

CBuilder 用のパッケージファイル

`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

5. Lazarus から

[File] → [Open] を選択して Packages\sgcWebSocketsLazarus.lpk を参照します（最初にコンパイルしてからインストールしてください）。

コンパイル済みファイルはLazarusディレクトリにあります。その中に、最新のIndyソースバージョンを含むIndyディレクトリがあります。

Lazarus 2.0.6およびIndy 10.5.9.4930でテスト済み

6. デモ

すべてのデモはDemosサブディレクトリで利用できます。プロジェクトを開いて実行するだけです。IntrawebデモはIntrawebのバージョンが異なるため、一部のユニットの修正が必要になる場合があります。

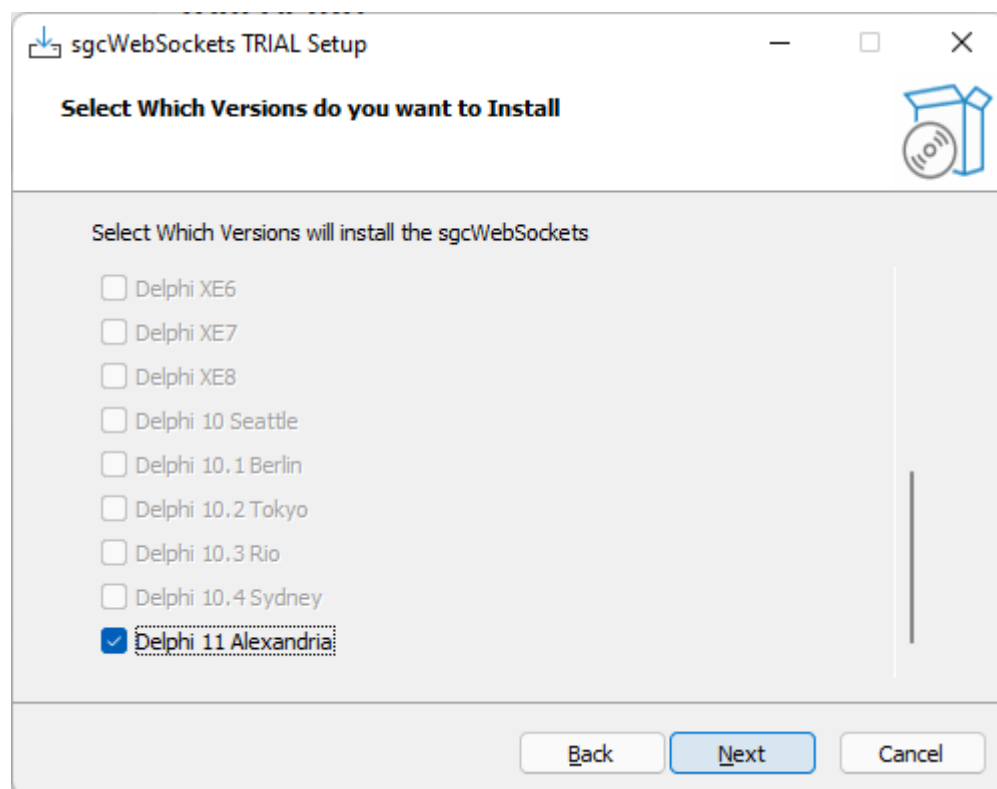
Install Setup

*最低でもWindows Vistaが必要です（Windows 2000、XP、Server 2003はサポートされていません）。

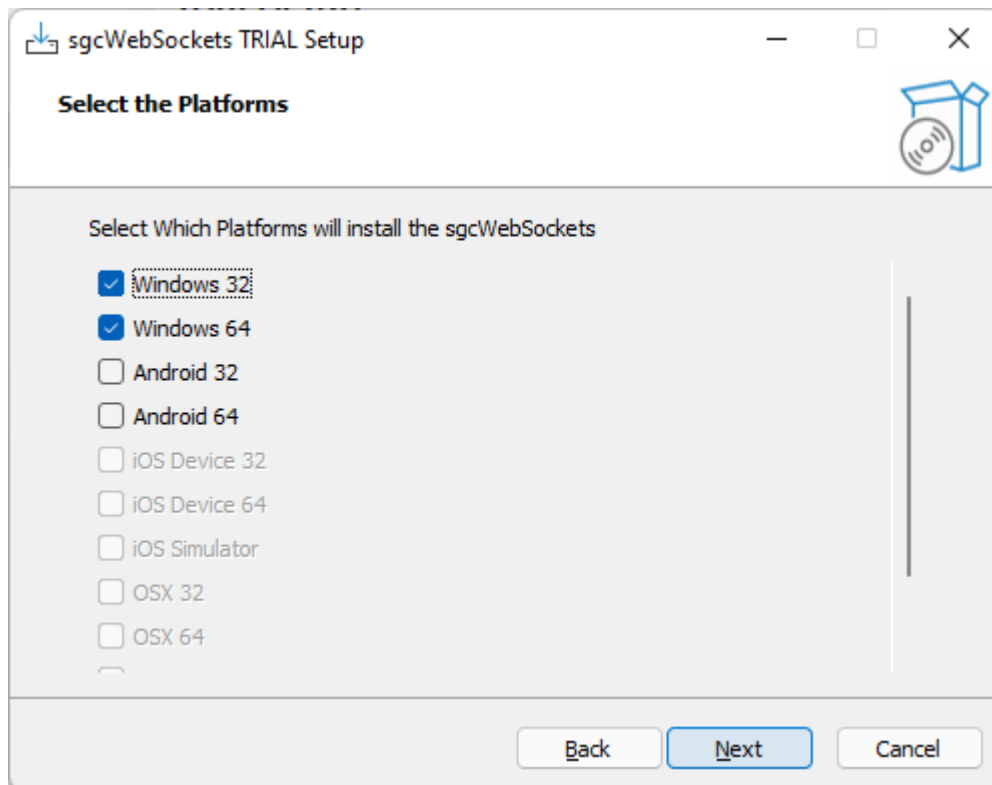
Windowsセットアップを使用してsgcWebSocketsライブラリをインストールする場合、インストールはガイド付きで非常にシンプルです。インストール中にエラーが発生した場合は、[インストールエラーページ](#)を参照するか、[パッケージを手動でインストール](#)することを試みてください。

試用版セットアップ

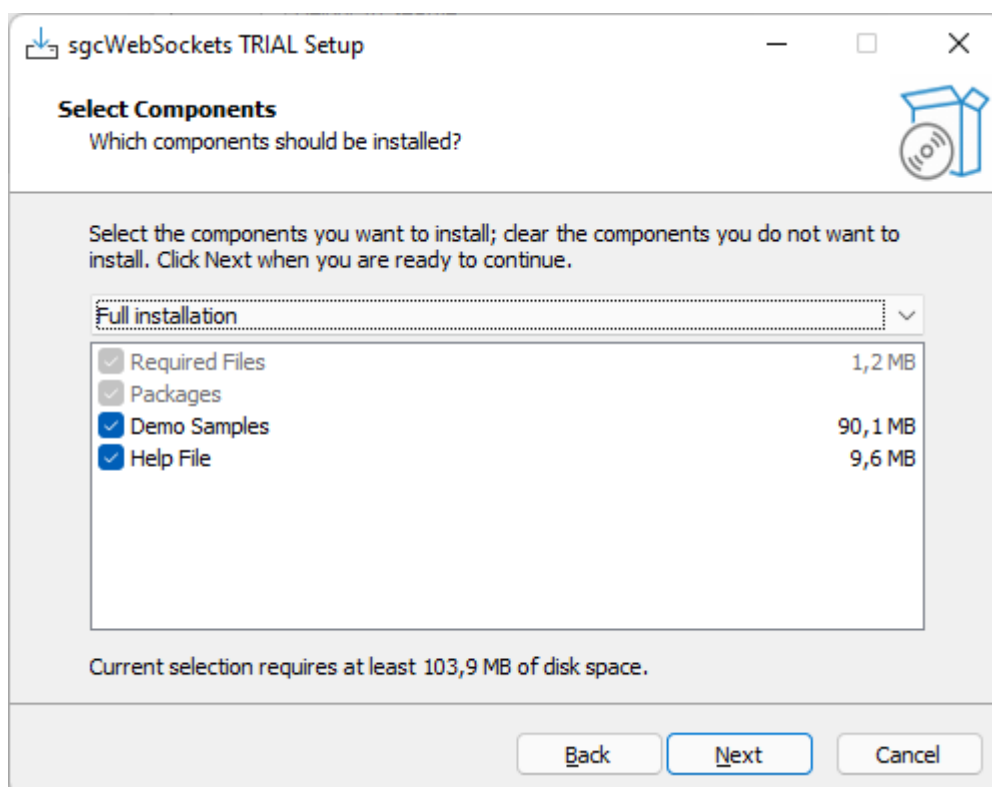
- トライアルインストーラーを実行します。
- トライアルのセットアップには管理者権限が必要です。
- インストーラーは Delphi / CBuilder / Rad Studio のバージョン一覧を表示し、デフォルトでダウンロードされたバージョンが有効になります。このバージョンがインストーラーによって検出されない場合、インストーラーはファイルを展開しますがコンパイルを試みません。[パッケージを手動でインストール](#)する方法を参照してください。



- 次のページには、インストールできるプラットフォームが表示されます。インストーラーで検出されたプラットフォームのみが有効になります。



- 次のページには、試用版のインストールに同意する必要があるライセンス契約が表示されます。
- ライセンス契約に同意した後、インストールされるコンポーネントが表示されます。デフォルトでは、パッケージ、コンパイル済み DCU、デモ、ヘルプファイルがすべてインストールされます。ヘルプファイルとデモをインストールするかどうかはカスタマイズできます。

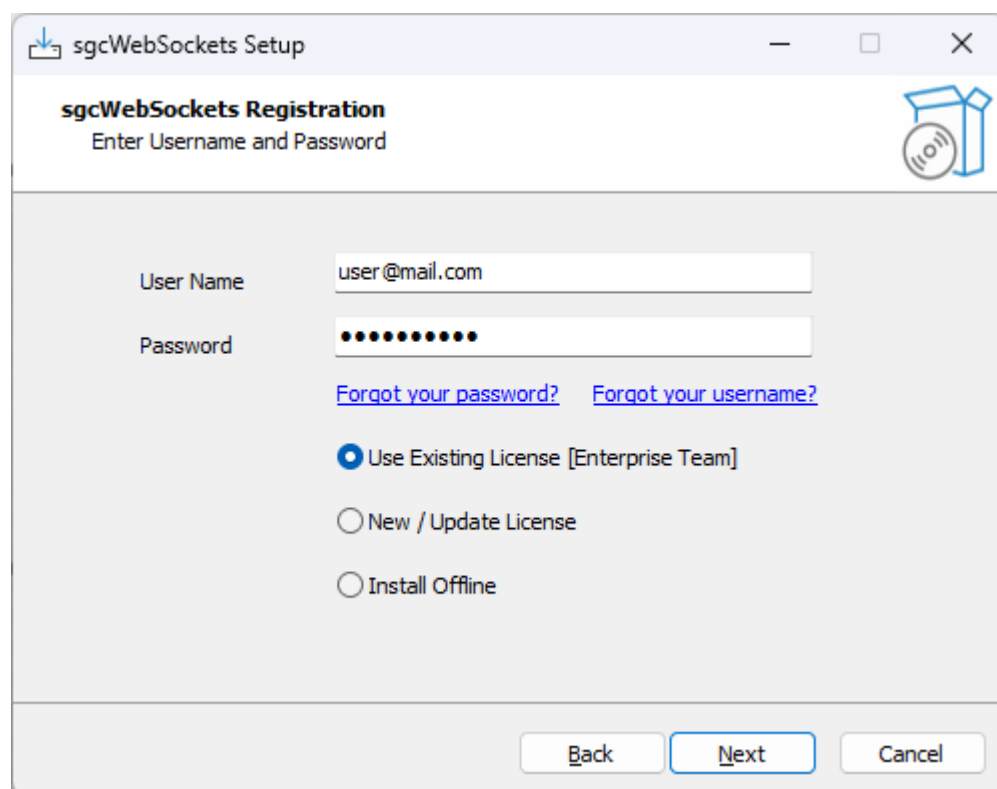


- 最後に、ファイルを展開し、パッケージをコンパイルしてインストールし、必要なパスを IDE に登録します。

顧客設定

ライセンスを購入したユーザーは、セットアップを使用して sgcWebSockets ライブラリをインストールできます。パッケージのインストール手順は以下を参照してください。

- インストーラーを実行します。
- インストーラーは最低限の権限で実行されます（管理者として実行するとネットワークドライブにインストールできません）。インストール先に管理者権限が必要な場合は、セットアップを管理者として実行してください。
- 最初に eSeGeCe プライベートアカウントのユーザー名/パスワードを設定する必要があります。これは一度だけ入力する必要があります。次回セットアップを使用する際、インストーラーは以前に保存された値を読み取ります。

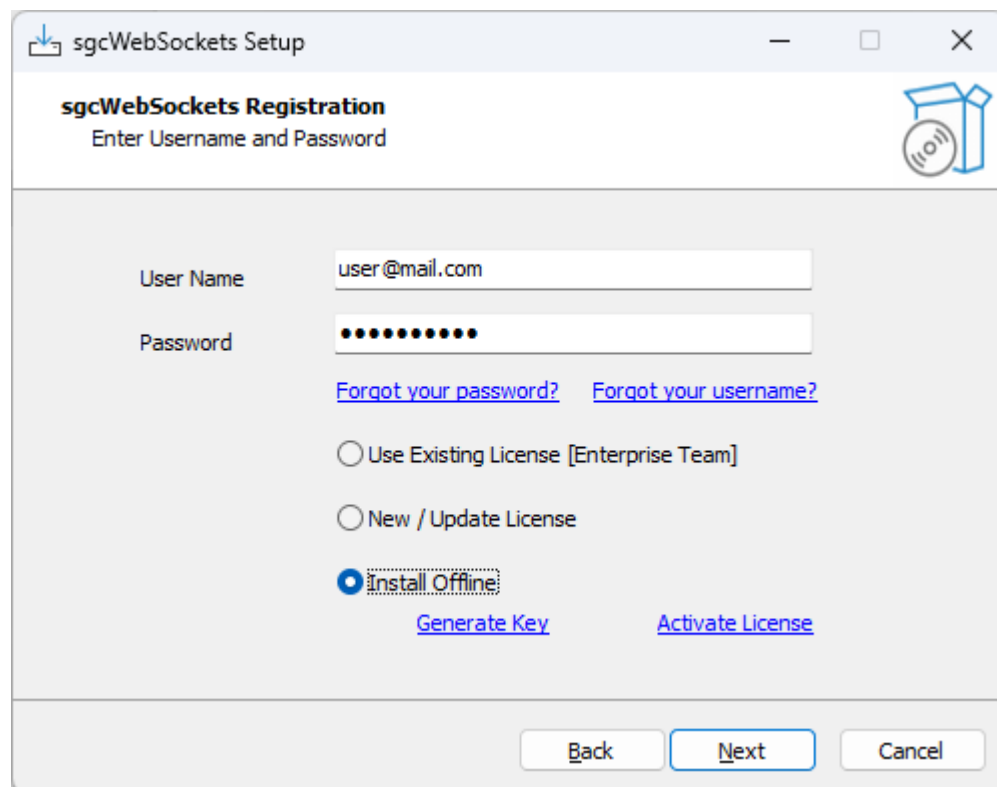


The image shows a screenshot of the 'sgcWebSockets Setup' window. The title bar says 'sgcWebSockets Setup'. The main heading is 'sgcWebSockets Registration' with the subtitle 'Enter Username and Password'. There is a small icon of a box with a coin on the right. The form has two input fields: 'User Name' 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 are three buttons: 'Back', 'Next' (which is highlighted), and 'Cancel'.

- 3つのオプションがあります:
 - **既存ライセンスを使用:** このバージョンがすでにインストールされている場合、このオプションがデフォルトで選択されます。このバージョンの最新の構成を使用します。
 - **新規 / ライセンス更新:** このバージョンが以前にインストールされていない場合、このオプションがデフォルトで選択されます。Server License に接続してライセンス情報を取得します。最近ライセンスをアップグレードした場合は、このオプションを選択してライセンスを更新できます。
 - **オフラインインストール:** マシンがインターネットにアクセスできない場合は、このオプションを選択してライセンスをアクティベートします。

キーの生成

このオプションは、ライセンスを有効化するために使用するキーを生成します。



sgcWebSockets Setup

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](#)

キーをコピーしてプライベートオンラインアカウントにアクセスします: www.esegece.com/my-account/subscriptions.



Generate Key - eSeGeCe Software (2023.8.0)

VjJ4b2MxTXlVWghpUm1oVltdHdZVlpXU2xOak1VNvIZMGhhYUZZd01UVlhMlIzVmpBeGMxZHVRbGRXUldzeFdWY3hSbVZXV25GUmJYQnNWbGQ0TmxVeFZrOVRNREZ6WTBac1ZHSnVRbTlXym5CeIRWWk9XR05JV21GTlIzaDRWVEZvYTFsdfJuSk5WRVpWVVFVd2QxcEVRbk5qUjFGNVdrVndhVll6YUhsVk1XUjNaR3h2ZDJKSvJ3sUlHSM2hvVld0YWQyTnNaRmhOVmtwT1RvaG9XVll4VWt0WlZscEdVbXBDVldFeFNsQlphMVV4VmtVeFdHskZO VmRTVlhCM1ZrVmFVMU5yTVZaTlNHeFRWa1phVUZsWE1VOUSNVkoxWTBoT2FWWlVSVEZWTVdoUfUyeEZkMkd6YUZwbGEzQlIxYbFprUzJJSR1Nu VIZiVvpXVVZhmIF3VFRGU2NsVnJTazITvkZaV1ZteG9hMVZHv2taU2FsSlZWbFpLZfZSc1ZYaFNWa1pWVld4Q1YxSXpVWHBXUkVaVFVXMVdjazFW Vm1GbGEwcFBXVmQwUmsxR1VYaFZiSEJyVfVSc1ZsVldVbGRVYkVwSVpFuk9WVTFHU2tOVWExVTFVbFpTV1ZwR1FsTlNNRFF4VmtaV1UySnJN WEpQVmxalU1ZrWktVRmxYTVU5Tk1WRjRWV3RHYTAxRVJrWlpWRTV2VkrKR2NsSlVSbFZXykVweFdrUkJOVlpXU2xWVGF6VIRVak5STUZaSE1Y ZFJheJXVfZWV1dGZEhVbEZWYTFwV1RWWlNWbFZ0TlU1aGvsWldWVlpTVTFReVJuUmtSRVWVfVas1lWUnNWWGhTVm5BMlUyeENVMUpWVl hkV1J6RjNzbTFSZDA5VlZsaFdsWEJQVld0a2FtYldVWGRWYTBwUfVsUlZnbfZzYUd0VlJrcElWR3BPVkJaNVVqTlZSa1U1VUZFOVBRPT0=

有効にするサブスクリプションを選択してキーを貼り付けます。

Offline License

TKVWSyprUK3VWVH10ZtOVWEXVH1V0rptV1ZWRK1SHNNK1F4vmtav10yShnNWLPQvma01Z1WVtVkmX
YTUVU5Tk1WRjRWV3RhYTAxRVJrWlpWRTV2VkrKR2NsSIVSbFZXyKVweFdrUKJOVlpXU2xWVGF6VIRVak5S
TUZaSE1YZFJhelZXVFZWV1dGEHvEbZWTfWv1RWWINWbFZ0TIU1aGVsWldWVlpTVTFReVJuUmtSRTV
WVfVaS1WUnNWWGhTVm5BMLUyeENVMUUpWVlhkV1J6RjNZbTFSZDA5VIZsaFdSWEJQVld0a2FtVldVblJqUI
VwUFVsUldRmxVVG10VJrcEIWR3BhVkJaNVqTIZSa1U1VUZFOVBRPT0=

ライセンスをアクティベートする

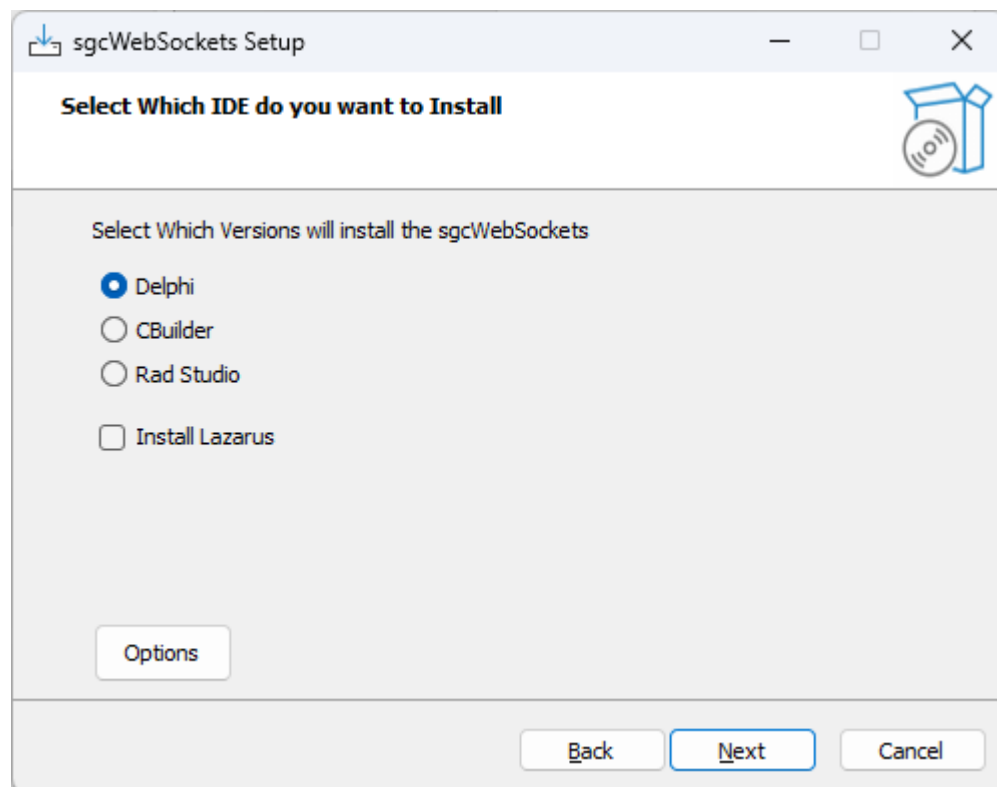
リクエストが正しければ、セットアップにコピーする必要があるライセンスが返されます。

Activate License - eSeGeCe Software (2023.8.0)

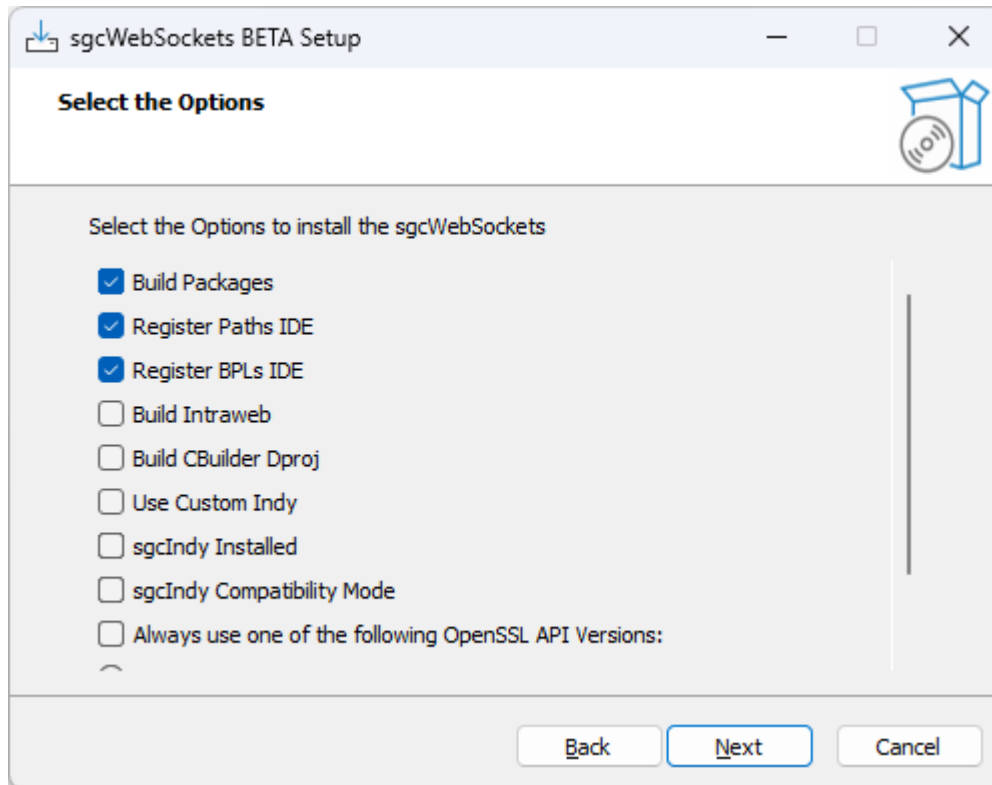
×

VmRTVlhCM1ZrVmFVMU5yTVZaTlNHeFRWa1phVUZsWE1VOU5NVkoxWTBoT2FWWWVSVSEZWTvdofUfUyeEZkMkl6YUzwbGEzQlXbFprUzJSR1Nu
VIZiVpXVfVkemVWVXhWazlUTURGWVZGaHNWmKp0ZUZOVMJuQkhZakZzVjFwRVRtdFdiWFEXvKRGb1UxTnRSbGxhUnpWVWZucFdSRmxWWk
U1bGJGSnhWRzFzVGSxSVFubFZNV1IzWkd4dmQySkUbfZotW1oVFZGVldTMIzHYkhGVFZGWNJvakJ3U1ZWc1pITIVWa1YzVW01Q1lWSjZSa2ha
ZWtwSFYwWldkRmR0Um1saE0wRjVWMWQ0VDFock5YSmlSVkpVvJBkb1VGbFhNWHBOUmXmKfkwWmFIRlpVmxawFdIqIBVMjFHV1ZwSE5WUldlbFpRV1RCYWRtVlZOVMhoUjNSVvVteHZNvII5ZUU5VGF6VnlZa1ZTVkZkSFVsQmFWM1J5VFZaU1ZsVnROVTVTVkZaV1ZWWVlNVMVJ0Vm5OU2Fr
NVZUVVpLVDFwVlZYaFNWbTk2WWvtVMUxSlZWWGRXUm1oM1ZtMVJkMDlWVmxWWFNFSIBWV3RrYTA1R1VYafZhmHBQVWxSV1JsZHFubXR
WUmtwR1RvaHdWVlpXU25WVWZVWVXhZMVpHVlZWck9WTINWbGw2VmtSQ1UxWnJNSGR0VlZaaFpXdGFVRnBYZEabGJGRjRWV3MxYTAXRVZ
USlpWRTV6Vkd4S1NGULVSBfZpUmtwRFZGukJOVlpXYTNwYVJubHNZbGhSTWxaR1ZsTmliVkyzVGxVmRTVlhCM1ZrVmFVMU5yTVZaTlNHeFRWa
1phVUZsWE1VOU5NVkoxWTBoT2FWWWVSVSEZWTvdofUfUyeEZkMkl6YUzwbGEzQlXbFprUzJSR1NuVIZiVpXVfVkemVWVXhWazlUTURGWVZGa
HNWmKp0ZUZOVMJuQkhZakZzVjFwRVRtdFdiWFEXvKRGb1UxTnRSbGxhUnpWVWZucFdSRmxWWkU1bGJGSnhWRzFzVGSxSVFubFZNV1IzWkd4dmQySkUbfZotW1oVFZGVldTMIzHYkhGVFZ
GWNJvakJ3U1ZWc1pITIVWa1YzVW01Q1lWSjZSa2haZWtwSFYwWldkRmR0Um1saE0wRjVWMWQ0VDFock5YSmlSVkpVvJBkb1VGbFhNWHBO
mxmKfkwWmFIRlpVmxawFdIqIBVMjFHV1ZwSE5WUldlbFpRV1RCYWRtVlZOVMhoUjNSVvVteHZNvII5ZUU5VGF6VnlZa1ZTVkZkSFVsQmFWM1J
5VFZaU1ZsVnROVTVTVkZaV1ZWWVlNVMVJ0Vm5OU2FrNVZUVVpLVDFwVlZYaFNWbTk2WWvtVMUxSlZWWGRXUm1oM1ZtMVJkMDlWVmxWWFN
FSIBWV3RrYTA1R1VYafZhmHBQVWxSV1JsZHFubXRWUmtwR1RvaHdWVlpXU25WVWZVWVXhZMVpHVlZWck9WTINWbGw2VmtSQ1UxWnJNSGR
0VlZaaFpXdGFVRnBYZEabGJGRjRWV3MxYTAXRVZUSlpWRTV6Vkd4S1NGULVSBfZpUmtwRFZGukJOVlpXYTNwYVJubHNZbGhSTWxaR1ZsTmli
VkyzVGxVmRTVlhCM1ZrVmFVMU5yTVZaTlNHeFRWa1phVUZsWE1VOU5NVkoxWTBoT2FWWWVSVSEZWTvdofUfUyeEZkMkl6YUzwbGEzQlXbFprU
zJSR1NuVIZiVpXVfVkemVWVXhWazlUTURGWVZGaHNWmKp0ZUZOVMJuQkhZakZzVjFwRVRtdFdiWFEXvKRGb1UxTnRSbGxhUnpWVWZucFdSR
mxWWkU1bGJGSnhWRzFzVGSxSVFubFZNV1IzWkd4dmQySkUbfZotW1oVFZGVldTMIzHYkhGVFZGWNJvakJ3U1ZWc1pITIVWa1YzVW01Q1lWS
jZSa2haZWtwSFYwWldkRmR0Um1saE0wRjVWMWQ0VDFock5YSmlSVkpVvJBkb1VGbFhNWHBOUmXmKfkwWmFIRlpVmxawFdIqIBVMjFHV1Z
wSE5WUldlbFpRV1RCYWRtVlZOVMhoUjNSVvVteHZNvII5ZUU5VGF6VnlZa1ZTVkZkSFVsQmFWM1J5VFZaU1ZsVnROVTVTVkZaV1ZWWVlNVMVJ0V
m5OU2FrNVZUVVpLVDFwVlZYaFNWbTk2WWvtVMUxSlZWWGRXUm1oM1ZtMVJkMDlWVmxWWFNFSIBWV3RrYTA1R1VYafZhmHBQVWxSV1JsZ
HFubXRWUmtwR1RvaHdWVlpXU25WVWZVWVXhZMVpHVlZWck9WTINWbGw2VmtSQ1UxWnJNSGR0VlZaaFpXdGFVRnBYZEabGJGRjRWV3MxY
TAXRVZUSlpWRTV6Vkd4S1NGULVSBfZpUmtwRFZGukJOVlpXYTNwYVJubHNZbGhSTWxaR1ZsTmliVkyzVGx

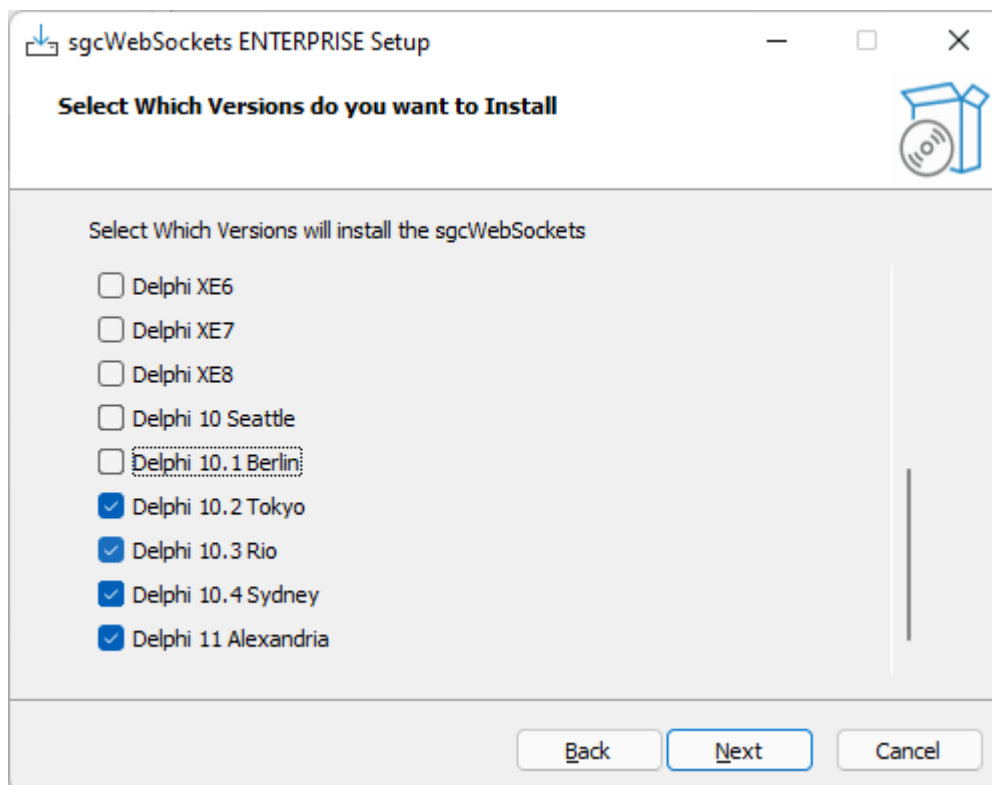
- ライセンスが正常にアクティベートされた場合、Delphi、CBuilder、またはRad Studio IDEのいずれにインストールするかを選択します。必要なLazarusファイルを抽出するためのチェックがあります（Lazarusではパッケージを手動でインストールする必要があります）。



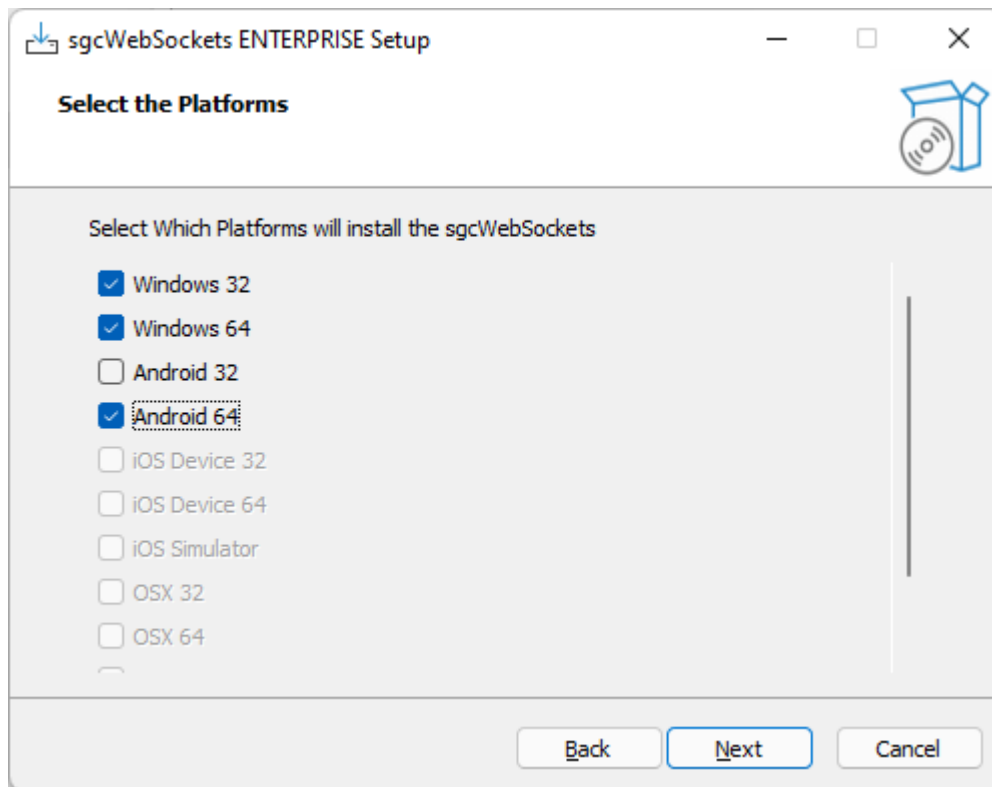
- インストーラーを使用するたびにカスタマイズできるオプションがあります。**オプション** ボタンを押してこれらのプロパティにアクセスします。
 - **パッケージのビルド:** 選択すると、インストーラーはパッケージのビルドを試みます。
 - **IDE パスの登録:** 選択すると、インストーラーは必要なライブラリパスを IDE に登録します。
 - **BPL の IDE への登録:** 選択し、インストーラーがパッケージを正常にビルドした場合、インストーラーはデザイン時パッケージを IDE に登録します。
- 以下のオプションはソースコードを含むライセンスでのみ利用可能です:
 - **Intraweb のビルド:** 選択すると、インストーラーは必要な Intraweb ファイルをインストールします（デフォルトでは無効）。現在 Intraweb XVI バージョンをインストールします。
 - **CBuilder Dproj のビルド:** 選択すると、インストーラーは sgcWebSockets Delphi パッケージを使用して CBuilder パッケージをビルドし、必要な CBuilder ファイルをすべて生成します。
 - **カスタム Indy の使用:** (Enterprise のみ) 選択すると、sgcWebSockets はカスタム Indy バージョン (openssl 1.1 および 3.0、TLS 1.3、ALPNなどをサポート) を使用します。
 - **sgcIndy がすでにインストール済み:** sgcIndy パッケージがインストールされており、このパッケージを使用して sgcWebSockets パッケージをコンパイルする場合は、このオプションをチェックしてください。
 - **sgcIndy 互換モード:** sgcIndy パッケージが互換モードでインストールされている場合 (DevExpress など他のパッケージが Indy を使用しているため)、このオプションをチェックしてください。
 - **常に以下の OpenSSL API バージョンを使用:** OpenSSL 1.1.1 または OpenSSL 3.0.0 API の使用を強制する場合はこのオプションをチェックしてください。
 - **デバッグモード:** ログファイルにデバッグメッセージを保存します。本番環境ではこのモードを使用しないでください。
 - **Rad Studio IDE Win64 のビルド:** Rad Studio IDE 12 以降の 64 ビット用デザインタイムパッケージをビルドします。
 - **リソースの含める:** デフォルトで有効です。JavaScript クライアントバンドル (sgcResources.RES) をランタイムパッケージに埋め込みます。埋め込まれたリソースは、サーバーが HTTP 経由でリモートブラウザに sgcWebSockets ブラウザクライアントを提供する場合にのみ必要です。このオプションのチェックを外すと、パッケージのコンパイル前に sgcVer.inc で SGC_RESOURCES が未定義になり、ソースにリンクするすべての実行可能ファイルのサイズを削減できます。Delphi / C++Builder クライアント専用アプリケーションや、バンドルされた JS クライアントを提供しないサーバーでは安全に無効にできます。基礎となるコンパイラディレクティブについては [インストールの設定](#) を参照してください。



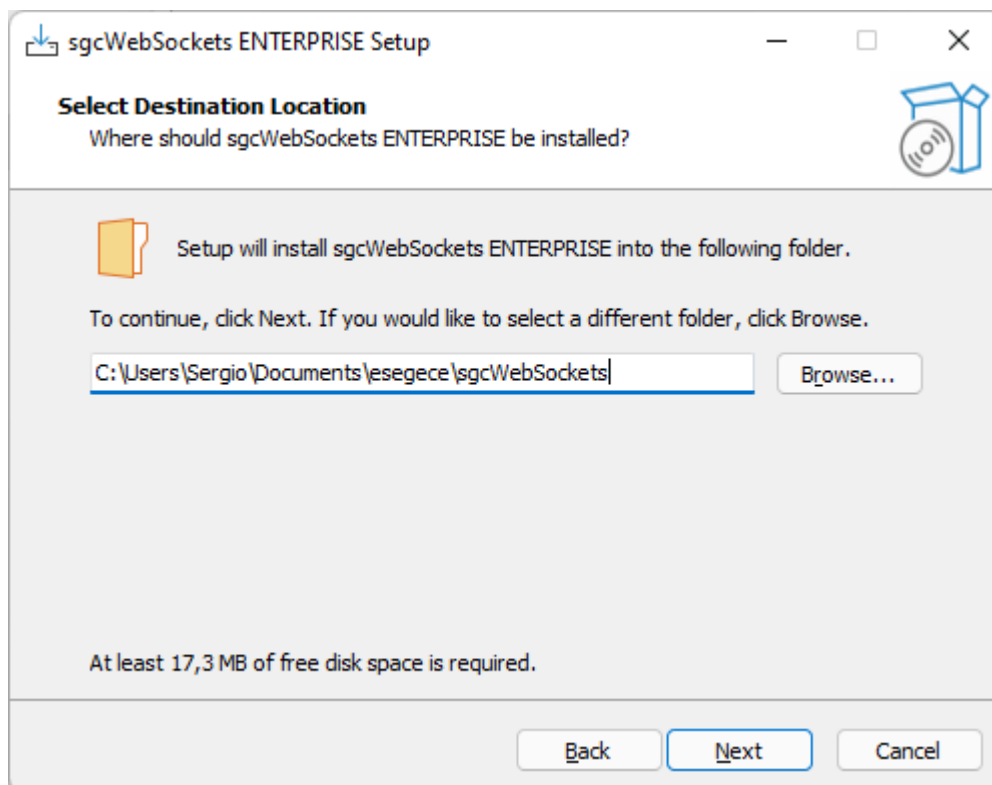
- インストールするIDEバージョンを選択できるようになりました。インストーラーがインストール済みとして検出したIDEバージョンのみが利用可能になります。



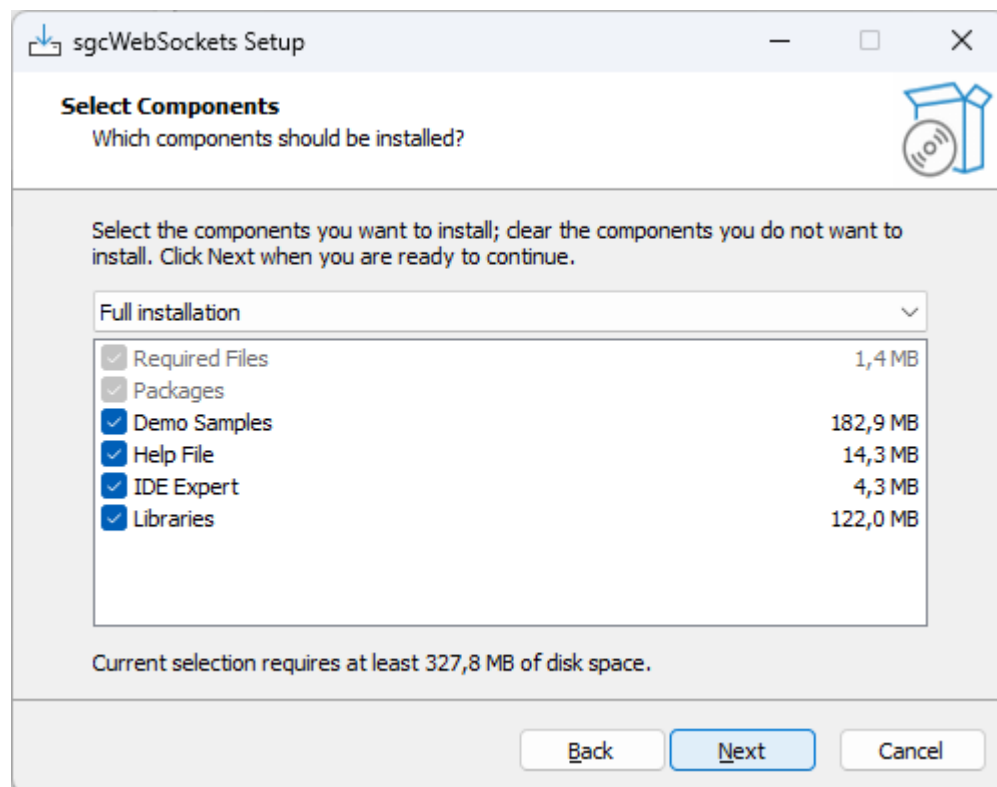
- 次のステップはプラットフォームを選択することです。



- パッケージをインストールするフォルダーを選択してください。パッケージを再インストールする場合、インストーラーはデフォルトで前回のインストールで選択したフォルダーを選択します。



- インストールするコンポーネントを選択します。登録済みのお客様には **IDE エキスパート**があり、IDE から eSeGeCe アカウントに接続したり、利用可能なアップデートを確認したり、ヘルプデスクに直接アクセスしたりすることができます。



- 最後に、ファイルを展開し、パッケージをコンパイルしてインストールし、必要なパスを IDE に登録します。

Install Errors

- MsBuild はライブラリパスが長すぎるとエラーを発生させます。この問題を解決するには、ライブラリパスから未使用のパスを削除してください。MsBuild には32K文字の制限があります。
- error **F2110: 'tchar.h' file not found**. このエラーはCBuilderでコンパイルする際に表示される場合があります。修正するには、CBuilderを開いて閉じてから再インストールを試みます。

インストールコマンドラインパラメータ

インストーラーは以下のコマンドをサポートしています。

/SILENT

ウィザードとバックグラウンドウィンドウは表示されませんが、インストールの進行状況は表示されます

/VERYSILENT

セットアップが非常にサイレントな場合、このインストール進行状況ウィンドウは表示されません。

/EXTRACT

パッケージはインストールされず、展開のみされます。インストールパスは /EXTRACT=path-to-folder でカスタマイズできます。

ユーザーの操作なしにファイルを展開するだけの場合は、このパラメータと /SILENT を使用します。

/IDE

このパラメータでは、インストールする IDE を設定できます。以下のいずれかを設定してください:

- delphi
- cbuilder
- radstudio

さらに Lazarus を追加することもできます。

使用例：delphi と lazarus をインストールします。

/ide=delphi-lazarus.

/VERSIONS

このパラメータを使用してインストールする RAD Studio バージョンを設定できます。複数のオプションが許可されています：

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

すべての可能なバージョンをインストールするには "All" の値を使用します。

例: Delphi 10 と Delphi 13 をインストールします。

/versions=D10-D13

/PLATFORMS

このパラメーターを使用して、インストールする RAD Studio プラットフォームを設定できます。複数のオプションを指定できます。

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

すべての可能なプラットフォームをインストールするには「All」という値を使用します。

例: Win32 と Win64 をインストールします。

/platforms=Win32-Win64

/USERNAME

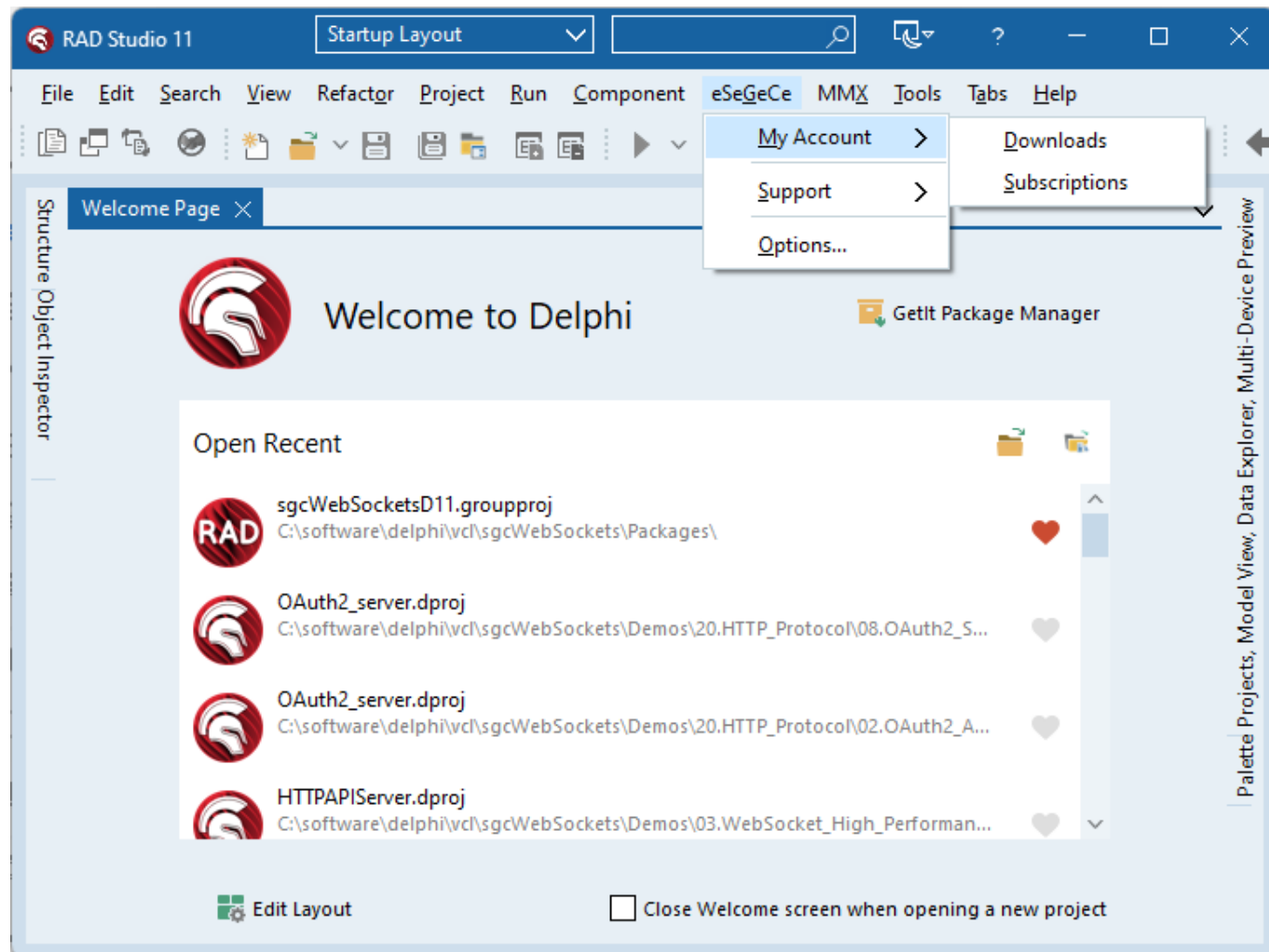
サブスクリプションのユーザー名を設定します

/PASSWORD

サブスクリプションのパスワードを設定します

IDE エキスパート

IDE エキスパートがインストールされている場合、以下のメニューオプションが表示されます。

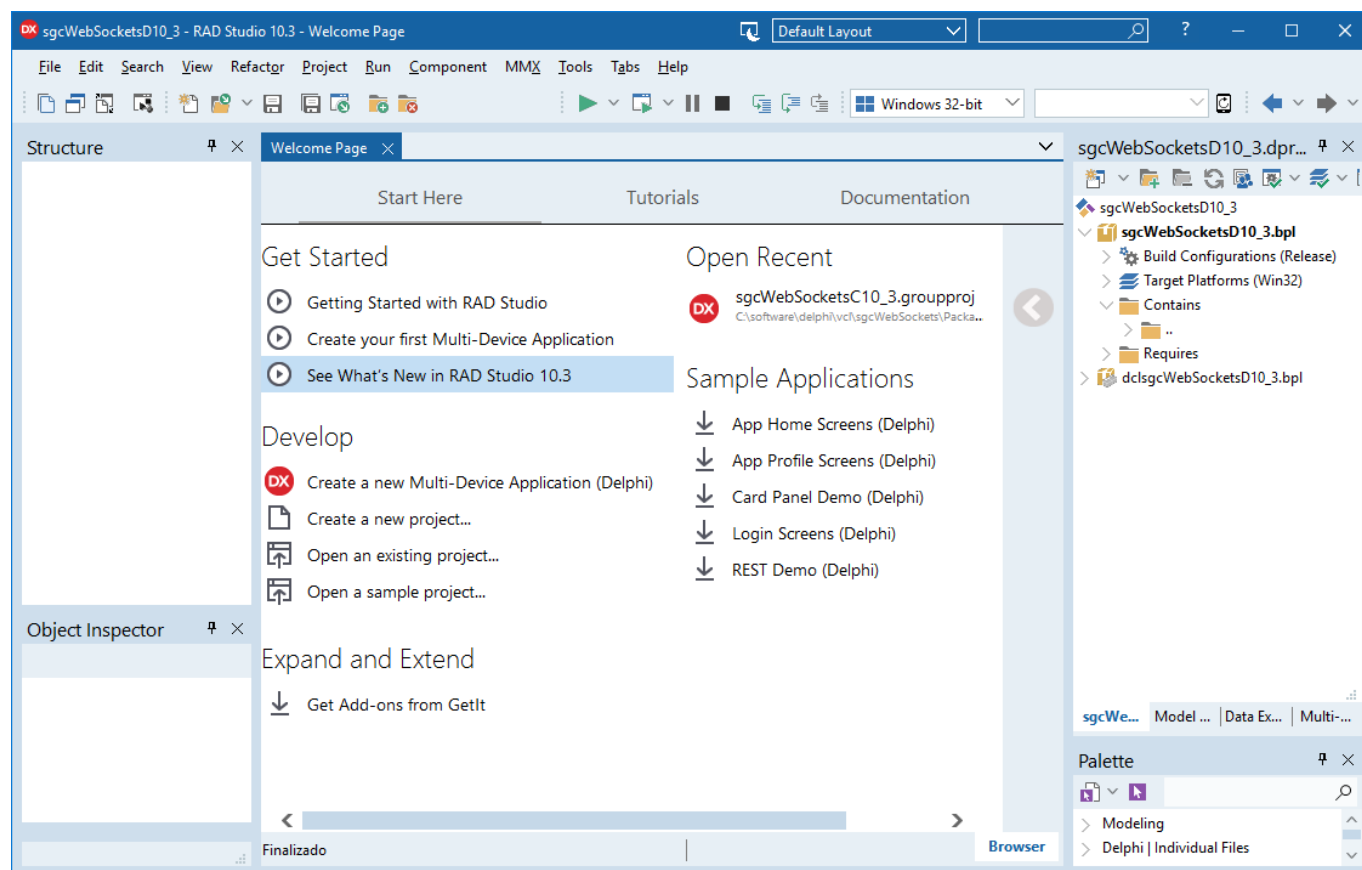


- **MyAccount:** ダウンロードメニュー (最新ベータ版をダウンロードできます) およびサブスクリプションへの直接アクセス。ライセンスの管理や期限切れライセンスの更新に使用します。
- **サポート:** 自動ログイン付きのヘルプデスクまたはフォーラムへの直接アクセス。ドキュメントとお問い合わせフォームも利用できます。
- **オプション:** このメニューではアカウントのユーザー名/パスワードを設定できます。デフォルトのブラウザを選択し、利用可能なアップデートがあるか確認します。

パッケージの手動インストール

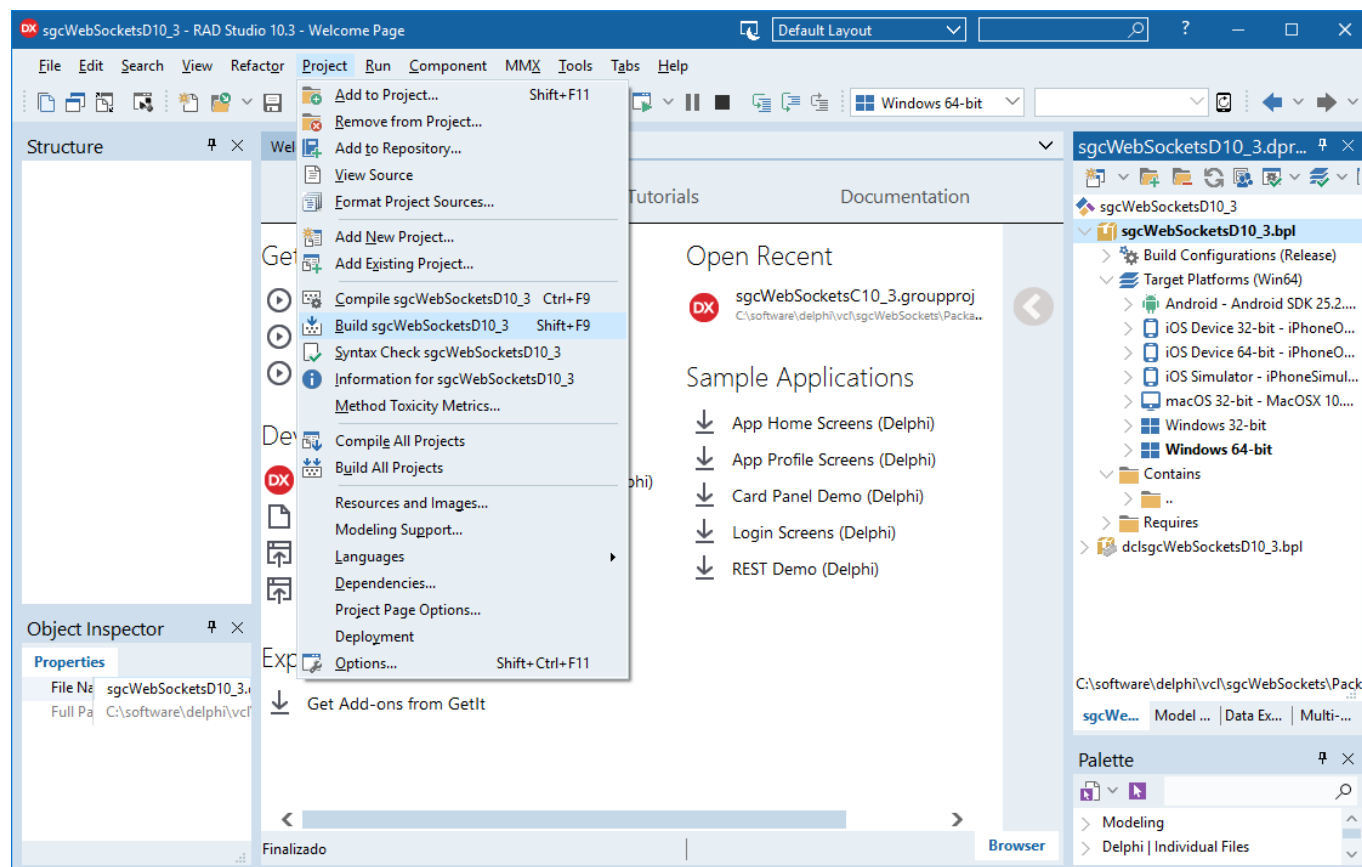
sgcWebSockets パッケージをインストールするには次の手順に従ってください。スクリーンショットは Delphi 10.3 バージョンを使用しています。

1. sgcWebSocketsD10_3 グループプロジェクトを開きます。

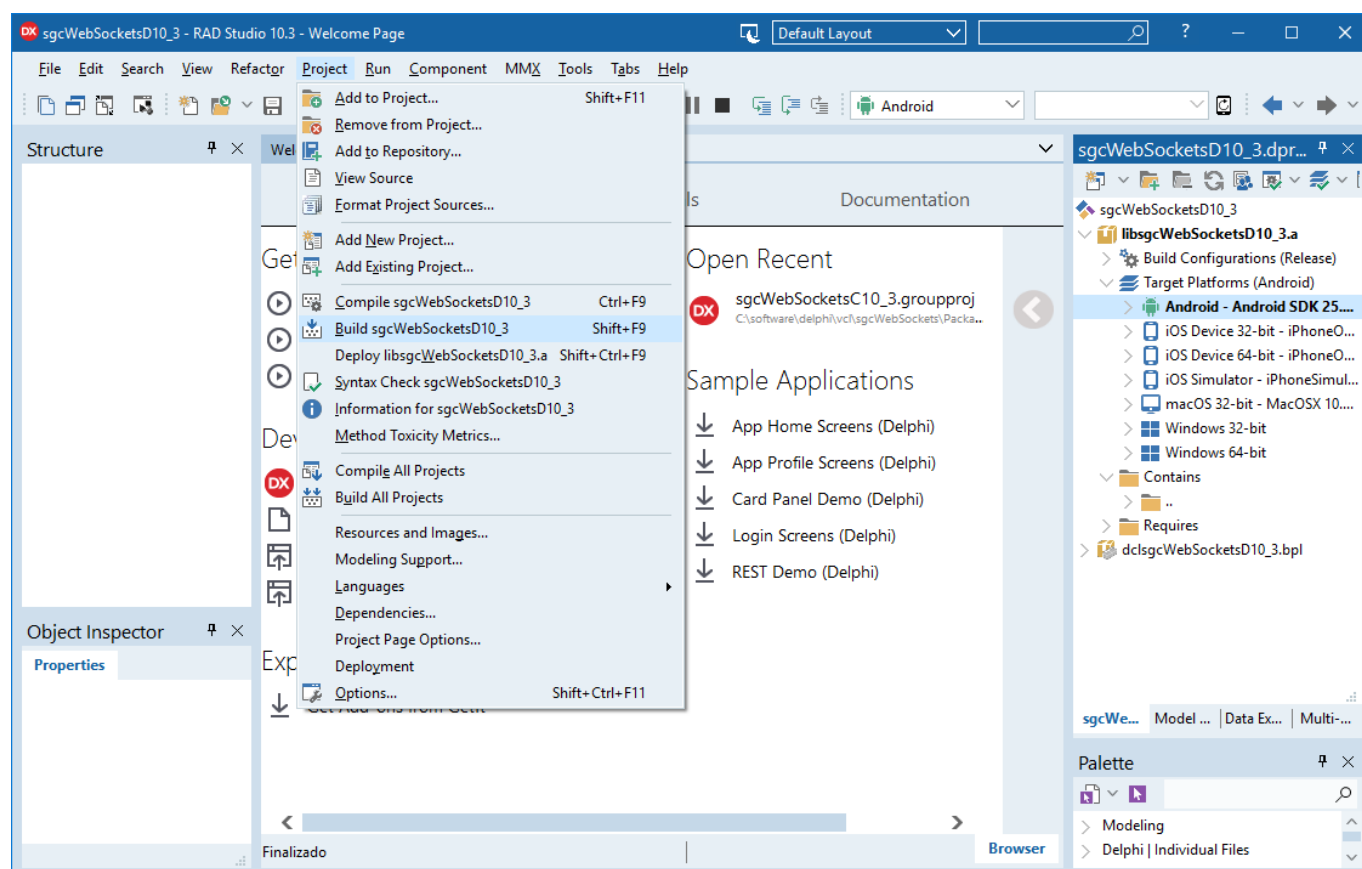


2. 次に **最初にランタイムパッケージをコンパイルする必要があります**（名前が sgcWebSockets で始まるもの）。ターゲットプラットフォームごとに 1 つのパッケージがあり、これは Delphi バージョンによって異なります。ターゲットプラットフォームを 1 つずつ選択して各パッケージをビルドしてください。

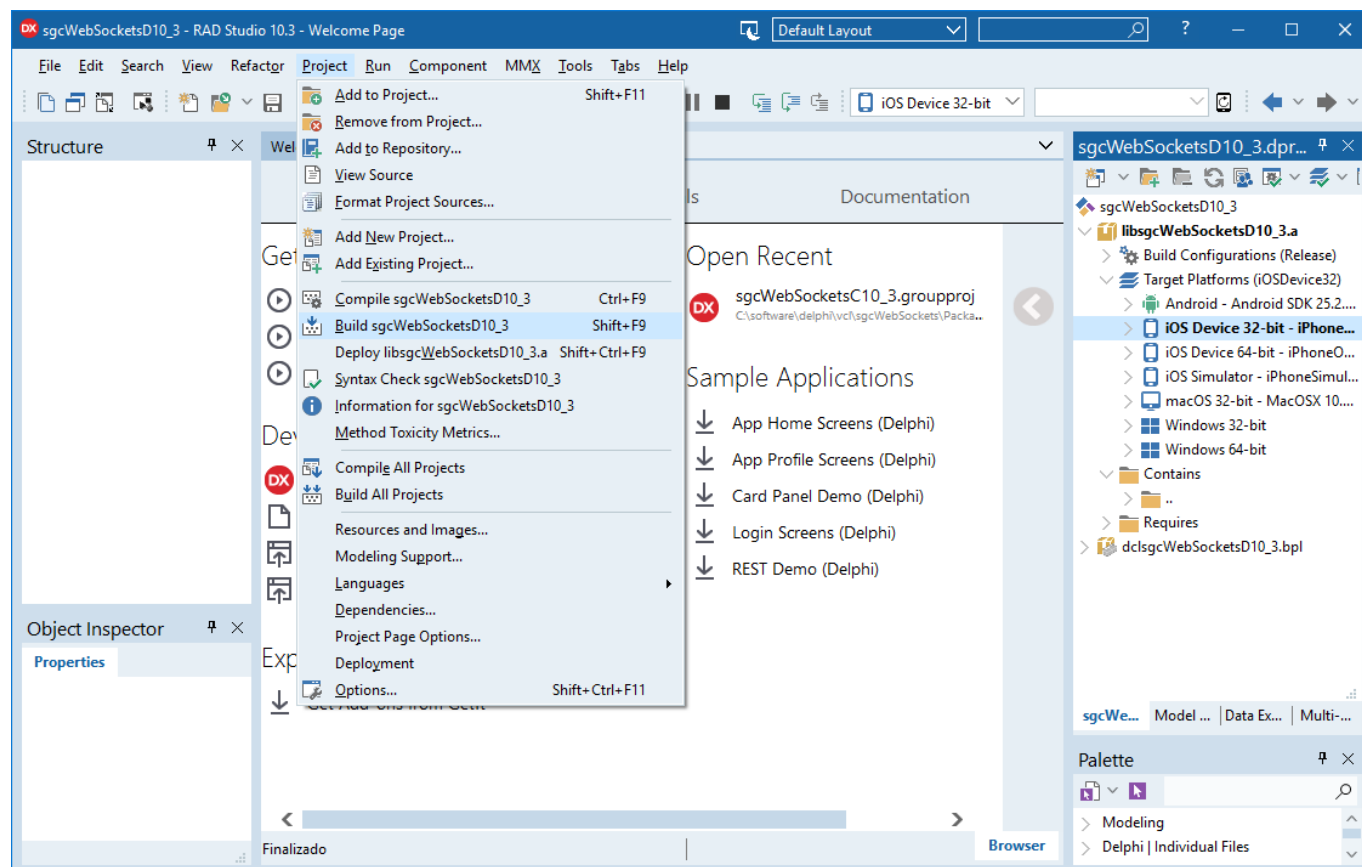
3. ターゲットプラットフォームとして win64 を選択してパッケージをビルドします。



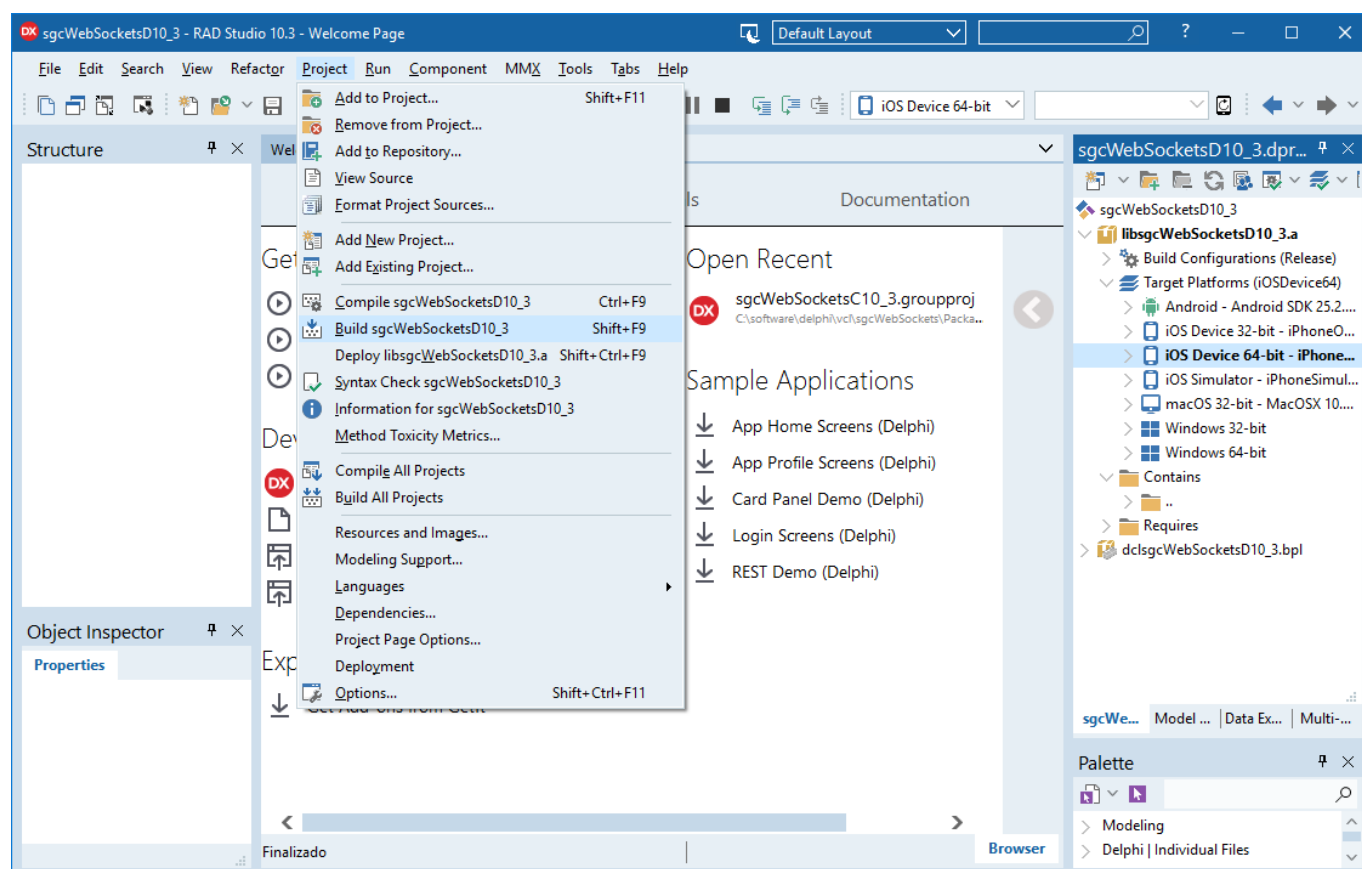
4. ターゲットプラットフォームとして Android を選択し、パッケージをビルドします。



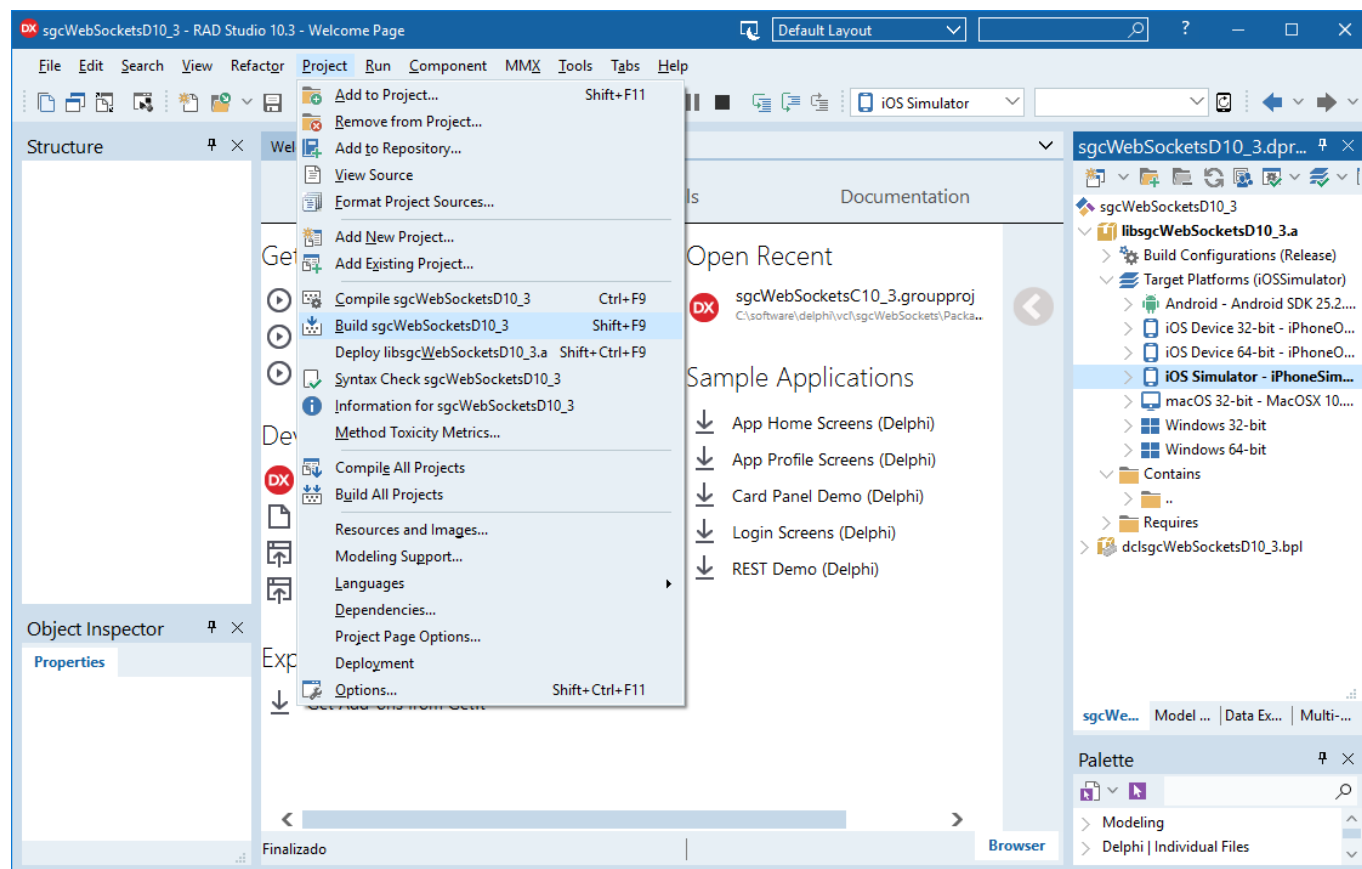
5. ターゲットプラットフォームとしてiOS Device 32 を選択し、パッケージをビルドします。



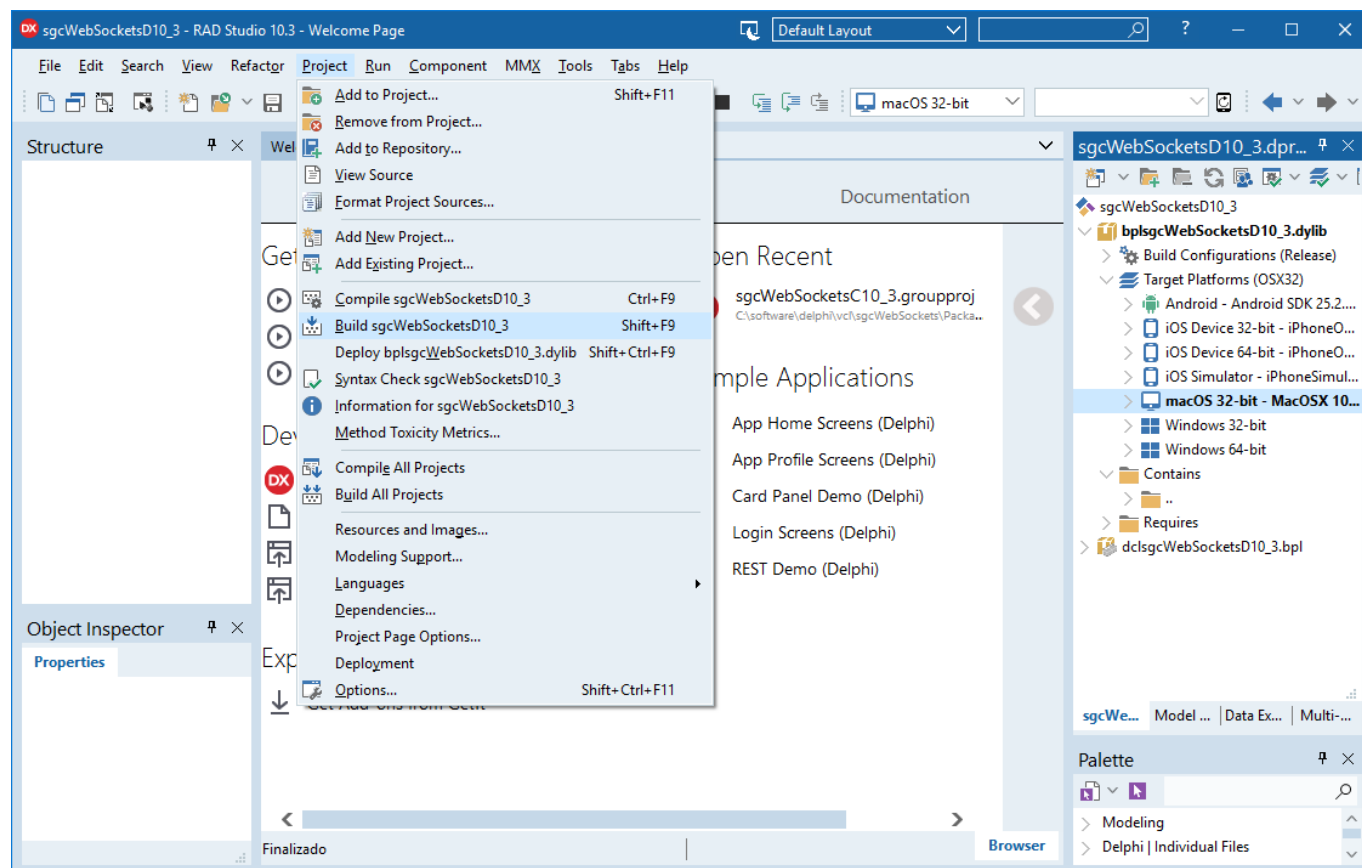
6. ターゲットプラットフォームとして **iOS Device 64** を選択し、パッケージをビルドします。



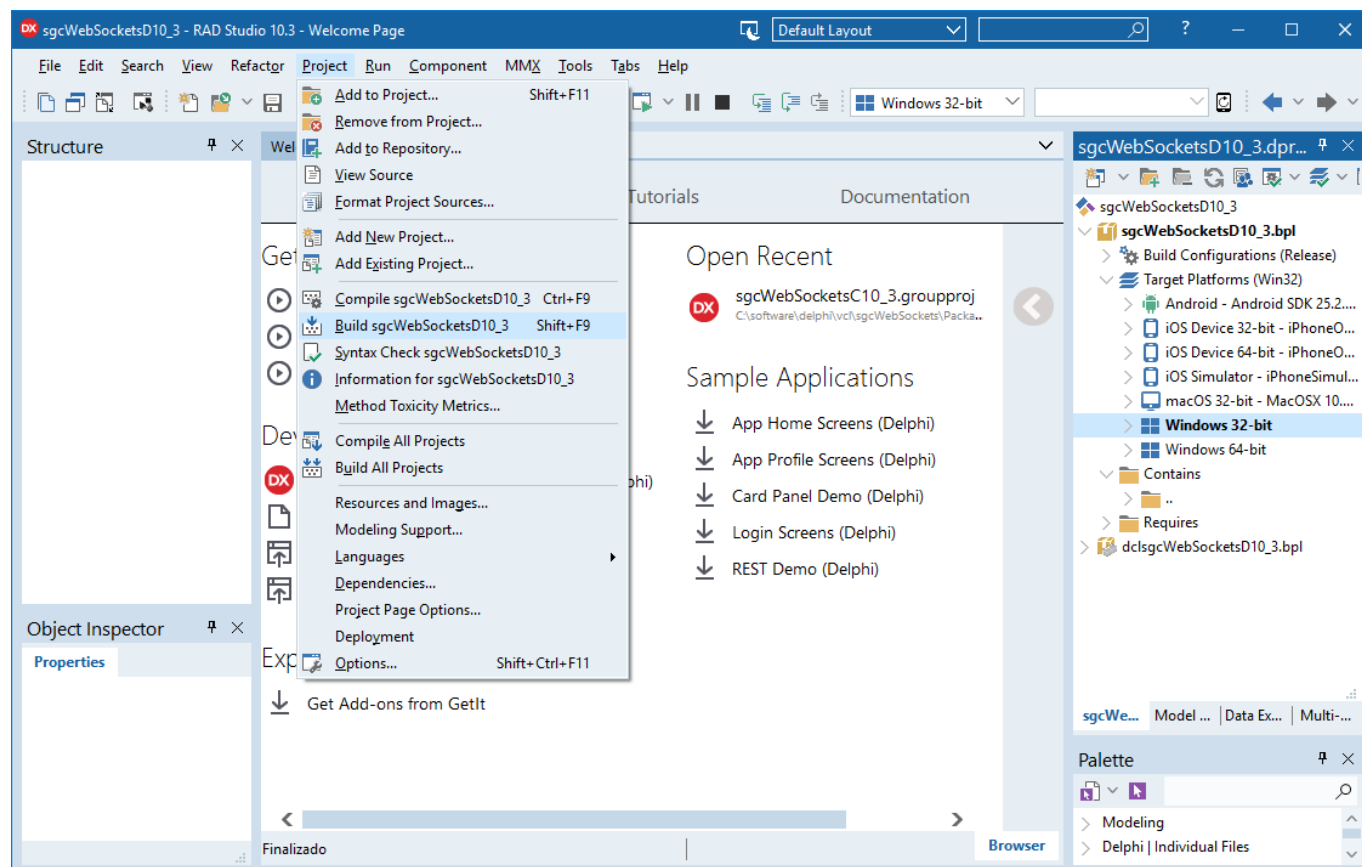
7. ターゲットプラットフォームとして **iOS デバイスシミュレーター** を選択し、パッケージをビルドします。



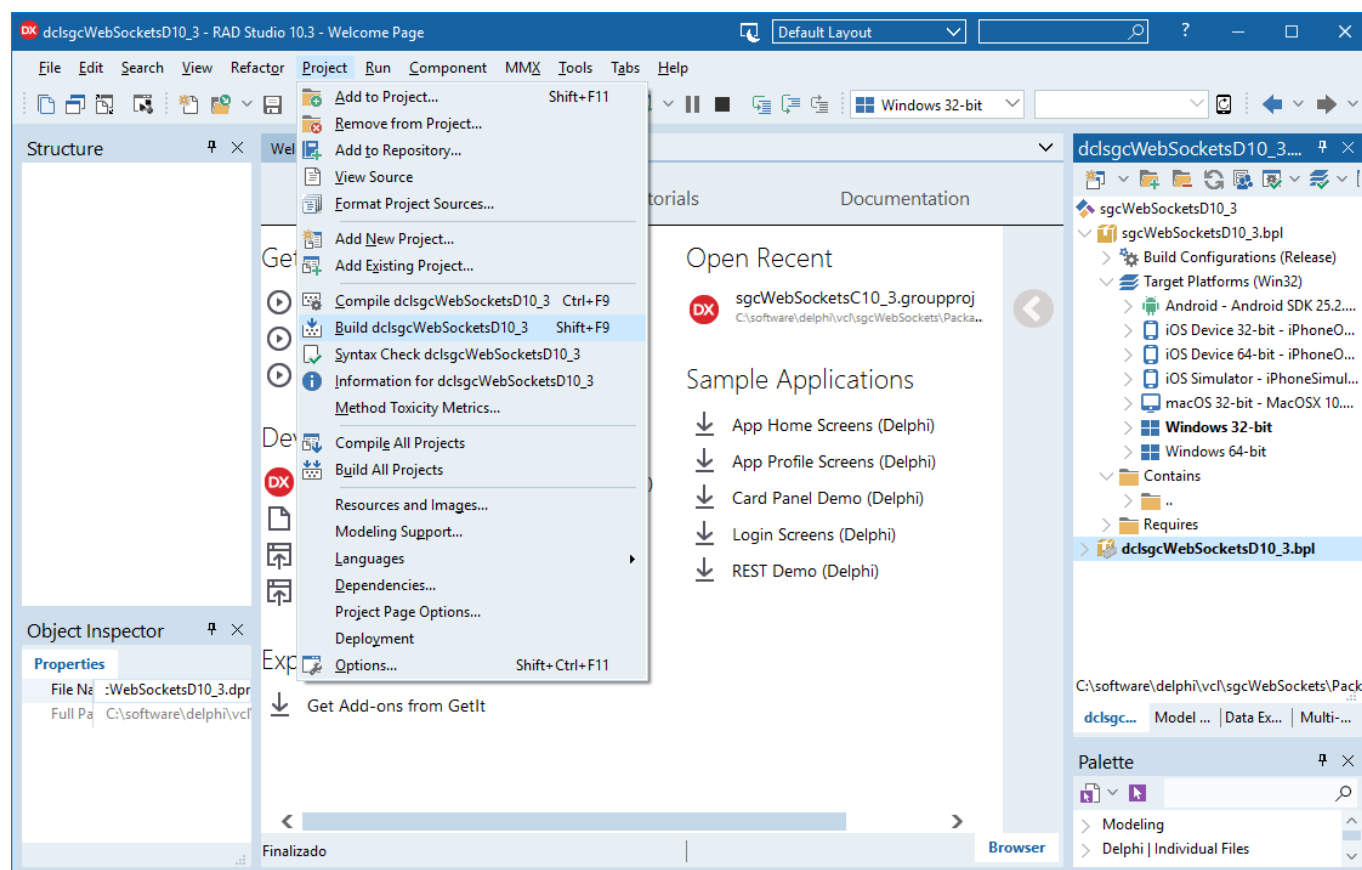
8. ターゲットプラットフォームとしてMacOS 32を選択し、パッケージをビルドします。

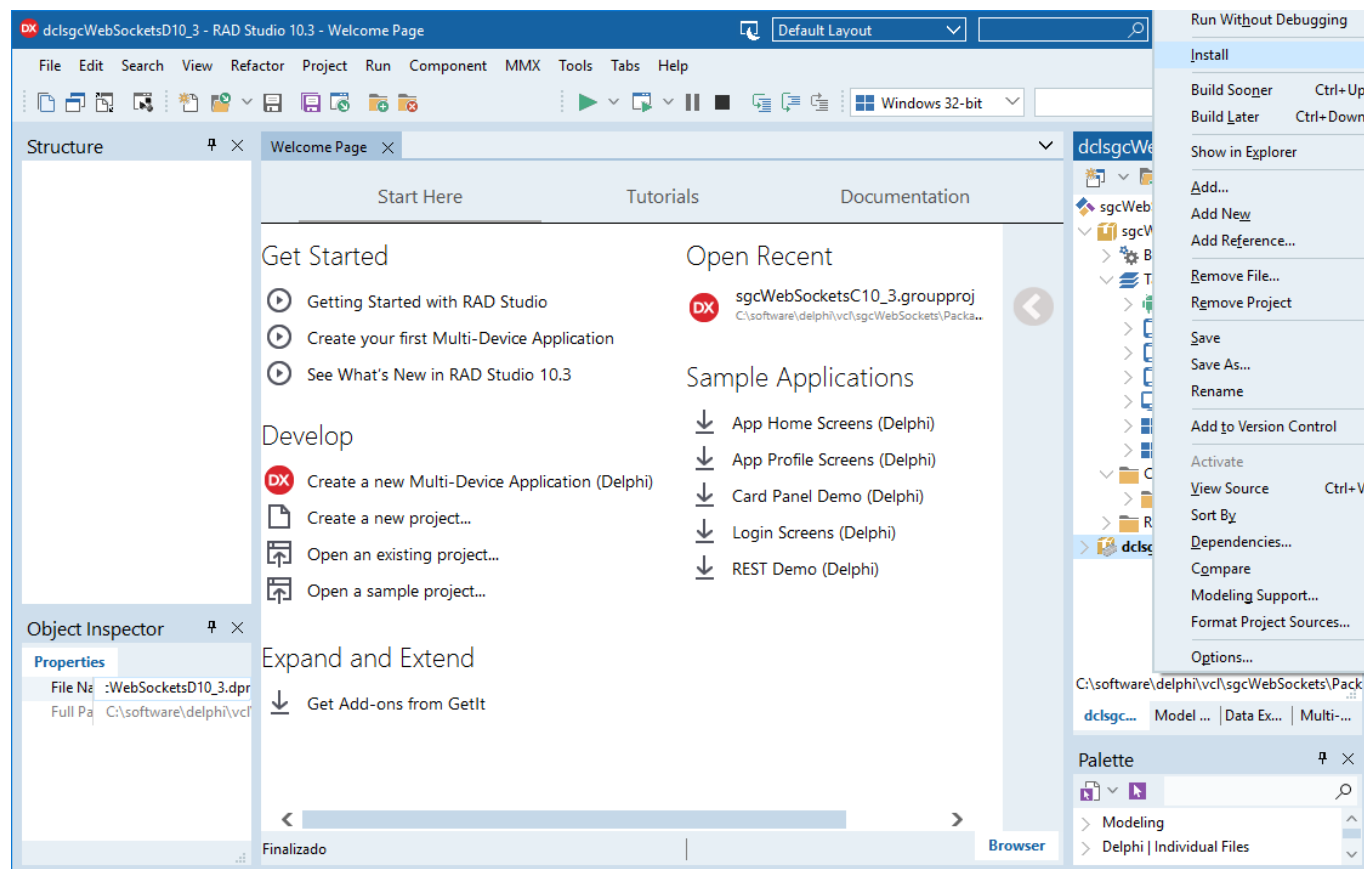


9. ターゲットプラットフォームとして Win32 を選択し、パッケージをビルドします。

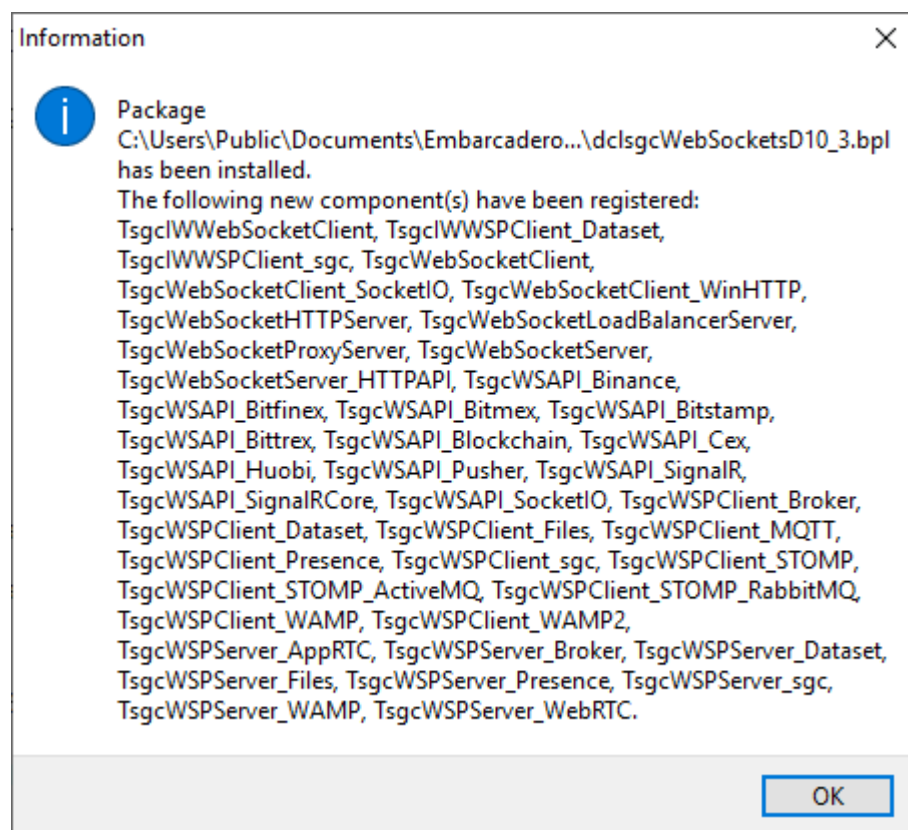


10. すべてのランタイムパッケージをコンパイルしたら、**デザインタイムパッケージ** (名前が dcl で始まる) を選択し、まずビルドしてからインストールします (デザインタイムパッケージは Win32 のみをターゲットプラットフォームとします)。

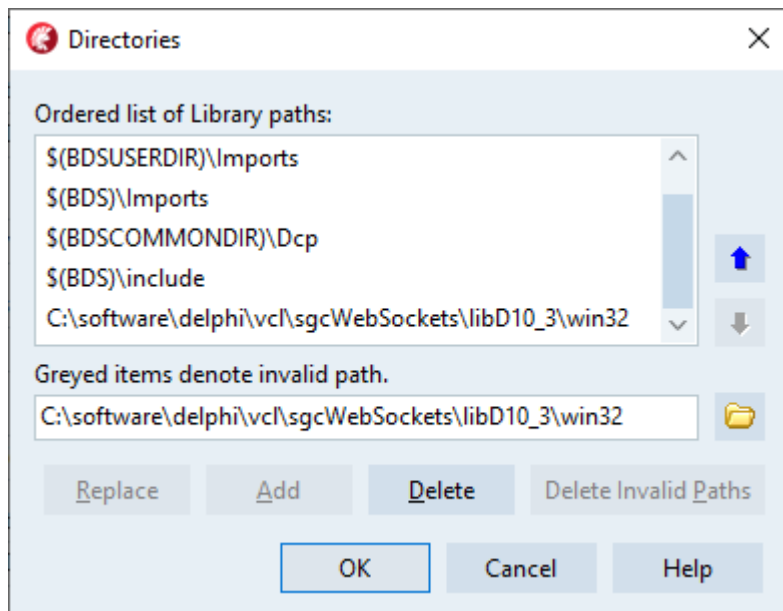




11. インストールが成功すると、すべてのコンポーネントがインストールされたというメッセージが表示されます。



12. 次に、コンパイル済みファイルが格納されているディレクトリを **Rad Studio ライブラリパス**に追加する必要があります。すべてのターゲットプラットフォーム（win32、win64、osx64...）に対してこれを追加する必要があります。



* **Datasnap** サーバーを使用している場合、これらはインストールできないため sgcWebSockets パッケージに含まれていません。ランタイム専用コンポーネントです。この場合、ライブラリパスに Source フォルダーも追加する必要があります。

Install Errors

コンポーネントのインストール時にエラーが発生する場合があります。

Intraweb package not found

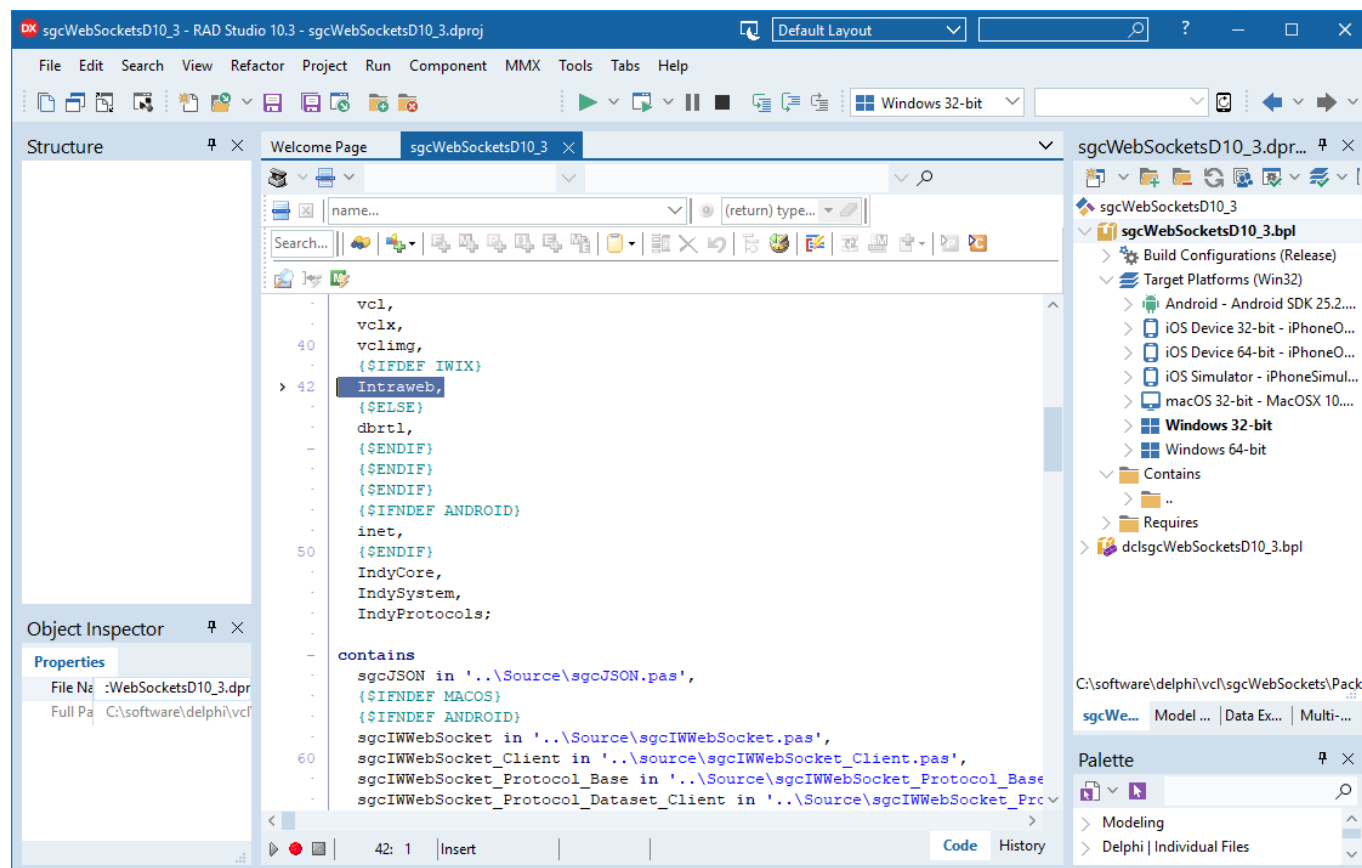
sgcWebSockets は Delphi に付属のデフォルトの **Intraweb** バージョンを使用してコンパイルされています。Intraweb がインストールされていない場合は、（Source フォルダーにある）**sgcVer.inc** ファイルを変更できます。

使用している Delphi バージョンを検索し、**Intraweb 用のすべてのコンパイラ定義をコメントアウトします**（IW で始まるもの）。例：Delphi 10.4 の場合、Intraweb のすべてのコンパイラ定義をコメントアウトします。

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

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

Intraweb がインストールされているものの、Delphi に付属するデフォルトとは異なるバージョンの場合、お使いの Intraweb パッケージの名前が異なる可能性があります。その場合は、sgcWebSockets ランタイムパッケージを開き、プロジェクトソース内で **Intraweb** 名を変更してください。



Indy パッケージが見つかりません

sgcWebSockets は IDE にコンポーネントをインストールするために **Indy** が必要です。トライアルのインストールは Delphi / CBuilder に付属の Indy ライブラリに対してコンパイルされているため、次のようなメッセージが表示された場合は:

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

おそらく新しい **Indy** バージョンをお持ちのため、トライアルをインストールするには、このバージョンを削除して Delphi / CBuilder セットアップを使用して組み込みの Indy バージョンをインストールする必要があります。

完全なソースコードをお持ちの場合は、以下を確認するだけで済みます:

1. 必要な Indy パッケージ: IndyCore、IndySystem、IndyProtocols。新しいバージョンの Indy を使用している場合、パッケージ名が異なる場合があります (バージョンを含む)。メニュー "**Component / Install Packages**" にアクセスし、Indy パッケージの名前を確認して、プロジェクトで適宜変更してください。

2. sgcWebSockets は複数の Indy バージョンをサポートしており、すべての Indy バージョン向けにコンパイルするためのコンパイラ定義があります。ソースフォルダーにある **sgcVer.inc** を開き、Indy バージョンに応じて変更します (IdVers.inc Indy ファイルの gslIdVersion)。いくつかのコンパイラ定義:

```
{ $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 (CBuilder)

CBuilder を使用してプロジェクトをコンパイルしてこのエラーが発生した場合、プロジェクトに次のオプションを設定してください：

プロジェクト > オプション > C++ リンカー
"Link with Dynamic RTL"のチェックを外す

Project > Options > Packages > Runtime Packages
「Link with runtime packages」のチェックを外す

CBuilder Win64でのUnable to find package import: sgcWebSocketsCXXX.bpi

win64 用のランタイムパッケージをコンパイルする場合は、Release と Debug の両方をコンパイルする必要があります。

System.ZLib.hpp と IdZLib.hpp の曖昧な参照 CBuilder

sgcWebSockets Standard および Professional は一部のコンポーネントにIndyを使用しており、IndyはZLibユニットを使用せず、IdZLib、IdZLibHeadersなどのZLibの独自コピーを使用します。プロジェクトはZLibとIndy ZLibユニットの両方にリンクしているため、コンパイル時に名前が同じであるため正しい参照がどれかわかりません。解決策は2つあります：

1. System.ZLib.hpp へのリンクが含まれている場所を検索し、削除するか IdZLibHeaders.hpp の後に移動してください
2. プロジェクトオプションで NO_USING_NAMESPACE_SYSTEM_ZLIB または DELPHIHEADER_NO_IMPLICIT_NAMESPACE_USE の条件定義を使用して System.Zlib.hpp の使用を回避します。

あいまいな参照 System.ZLib.hpp と sgclZLib.hpp CBuilder

sgcWebSockets Enterprise は一部のコンポーネントにカスタム Indy バージョンを使用しており、Indy は ZLib ユニットを使用せず、独自の ZLib コピー (sgclZLib、sgclZLibHeaders など) を使用します。プロジェクトは ZLib と Indy ZLib ユニットの両方にリンクしているため、コンパイル時に名前が同一であるため、コンパイラはどちらが正しい参照なのかを判断できません。解決策は2つあります。

1. System.ZLib.hppへのリンクが含まれている場所を検索し、削除するかsgclZLibHeaders.hppの後に移動します
2. プロジェクトオプションで NO_USING_NAMESPACE_SYSTEM_ZLIB または DELPHIHEADER_NO_IMPLICIT_NAMESPACE_USE の条件定義を使用して System.Zlib.hpp の使用を回避します。

CBuilder および Android で Sgcwebsocket... の vTable への未定義の参照

このエラーを修正するには次の回避策を使用してください。メニュー「プロジェクト / プロジェクトに追加」を使用して、dcp/android デフォルトフォルダーにある libsgcwebsocketsC*.a ファイルをプロジェクトに追加してください。

例：CBuilder 11 の場合、プロジェクトにファイル「libsgcWebSocketsC11.a」を追加します。このファイルはデフォルトで「C:\Users\Public\Documents\Embarcadero\Studio\22.0\DCP\Android\Release」フォルダーにあります。

Lazarus でチェックサムが変更されました

このエラーは、IDE をビルドするプロファイルが「Optimized IDE」でない場合に Lazarus でコンポーネントをインストールしようとしたときに発生することがあります。トライアルは「Optimized IDE」プロファイルでコンパイルされています。

Y が使用する X が見つかりません。互換性のない ppu

以下の回避策を試してくださいメニューオプションから「実行 / クリーンアップと再ビルド」。

Configure Install

ソースフォルダーには sgcVer.inc というファイルがあり、すべての Delphi、CBuilder、Lazarus IDE のコンパイラ定義が含まれています。

ここでIntraWeb、Indyなどの設定をカスタマイズできます。**通常は変更不要です**。一部の機能を有効/無効にしたい場合を除きます。

コンパイラの定義を慎重に変更し、サポートが必要な場合はお問い合わせください。

Delphi の各バージョンについて、すべてのコンパイラ定義を設定できるセクションがあります。例えば Delphi 10.4:

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

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

Indy

Indy ライブラリにはいくつかのコンパイラ定義があります。これはインストールされている Indy バージョンによって異なり、デフォルトでは Delphi に含まれている Indy パッケージ用に設定されています。Indy バージョンは IdVers.inc Indy ファイルの gsldVersion パラメータです。

Intraweb

Intraweb がインストールされていない場合は、Intraweb のコンパイラ定義（IW... で始まるもの）をコメントアウトしてください。

SGC_RESOURCES

デフォルトでは sgcVer.inc は {\$DEFINE SGC_RESOURCES} を宣言し、ランタイムパッケージにリソースファイル sgcResources.RES を埋め込みます。これにより、sgcWebSockets ソースに対して静的にリンクするすべての実行ファイルに余分なバイトが追加されます。

このリソースには、sgcWebSockets サーバーがリモートブラウザへ HTTP を通じて配信できるバンドルされた JavaScript クライアントコードが含まれています（ブラウザが sgc サブプロトコルを話せるように）。次の場合にのみ必要です：

- JSクライアントバンドルをブラウザに配信するsgcWebSockets HTTP/WebSocketサーバーを使用します（TsgcWSProtocol_JS_*ベースのサーバーで一般的です）。

次の場合は不要です：

- Delphi / C++Builder WebSocket クライアント（データはブラウザではなくネイティブコードで消費されます）。
- バンドルされた JS クライアントを提供しないサーバー（例：WebSocket 専用サーバー、ネイティブ間サーバー）。

埋め込みリソースを除外するには、sgcVer.incの行をコメントアウトします。

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

または、ソースエディションの [インストールセットアップ](#)を実行する場合、オプションページで **リソースを含める**オプションのチェックを外してください。インストーラーはパッケージをコンパイルする前に sgcVer.inc を書き換えます。

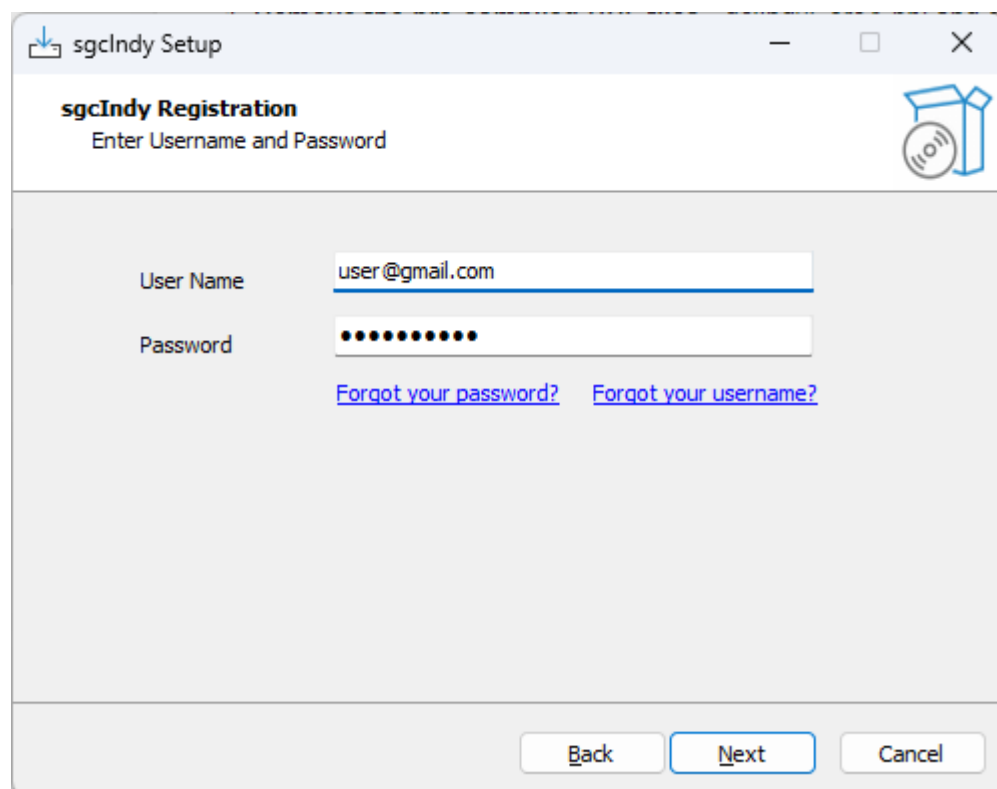
sgclndy パッケージのインストール

セットアップインストール

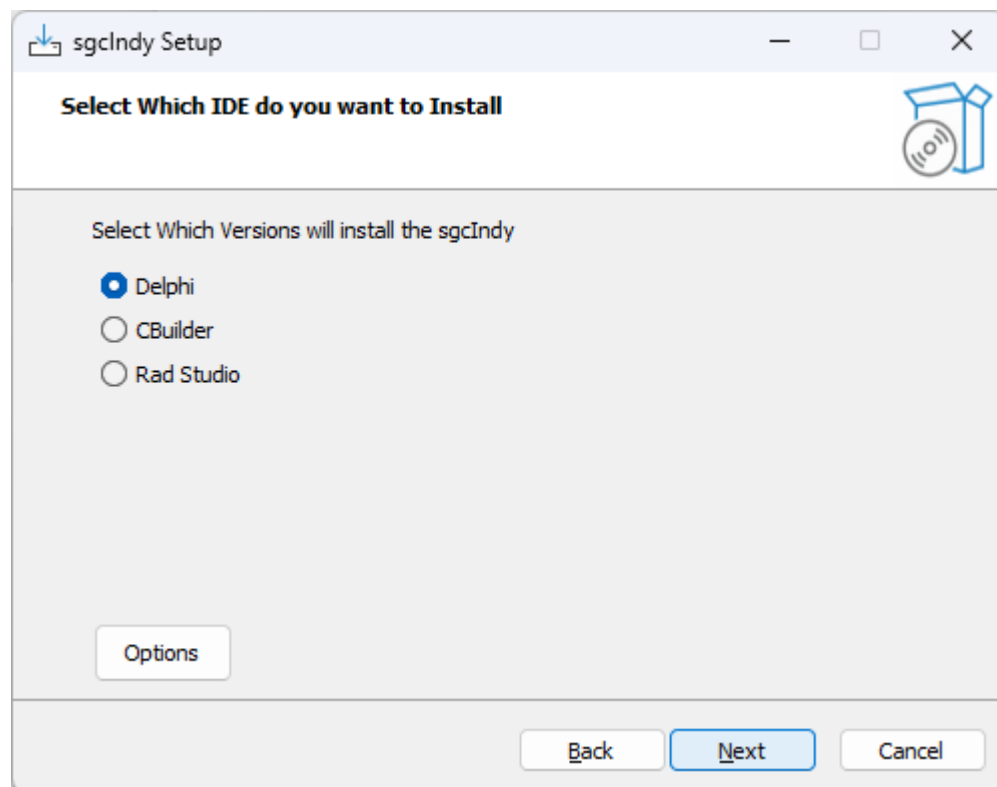
*最低でもWindows Vistaが必要です（Windows 2000、XP、Server 2003はサポートされていません）。

ライセンスを購入したユーザーは、セットアップを使用して sgclndy パッケージをインストールできます。以下にパッケージのインストール手順をステップごとに示します。

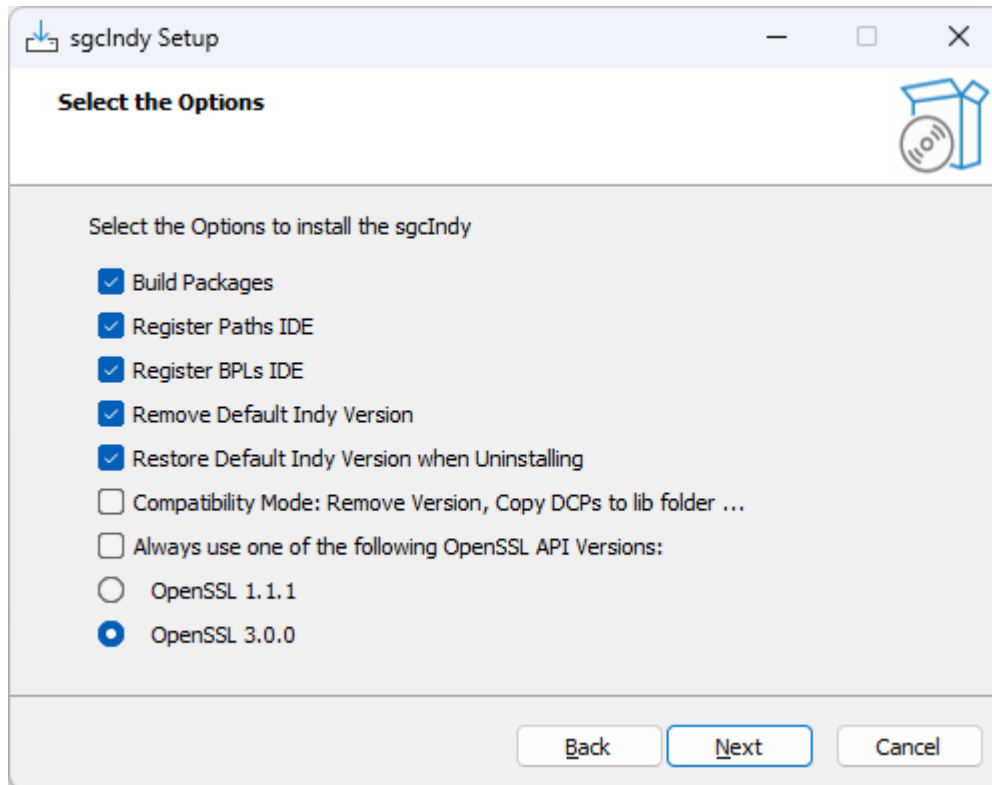
- インストーラーを実行します。
- まずプライベートな eSeGeCe アカウントのユーザー名/パスワードを設定する必要があります。これは一度だけ入力すればよく、次にセットアップを使用するときにインストーラーが最新の値を読み取ります。

The image shows a screenshot of the 'sgclndy Setup' window. The title bar says 'sgclndy Setup'. The main content area is titled 'sgclndy Registration' with the subtitle 'Enter Username and Password'. There is a small icon of a box with a dollar sign in the top right corner. Below the title, 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, there are two links: 'Forgot your password?' and 'Forgot your username?'. At the bottom of the window, there are three buttons: 'Back', 'Next', and 'Cancel'. The 'Next' button is highlighted with a blue border.

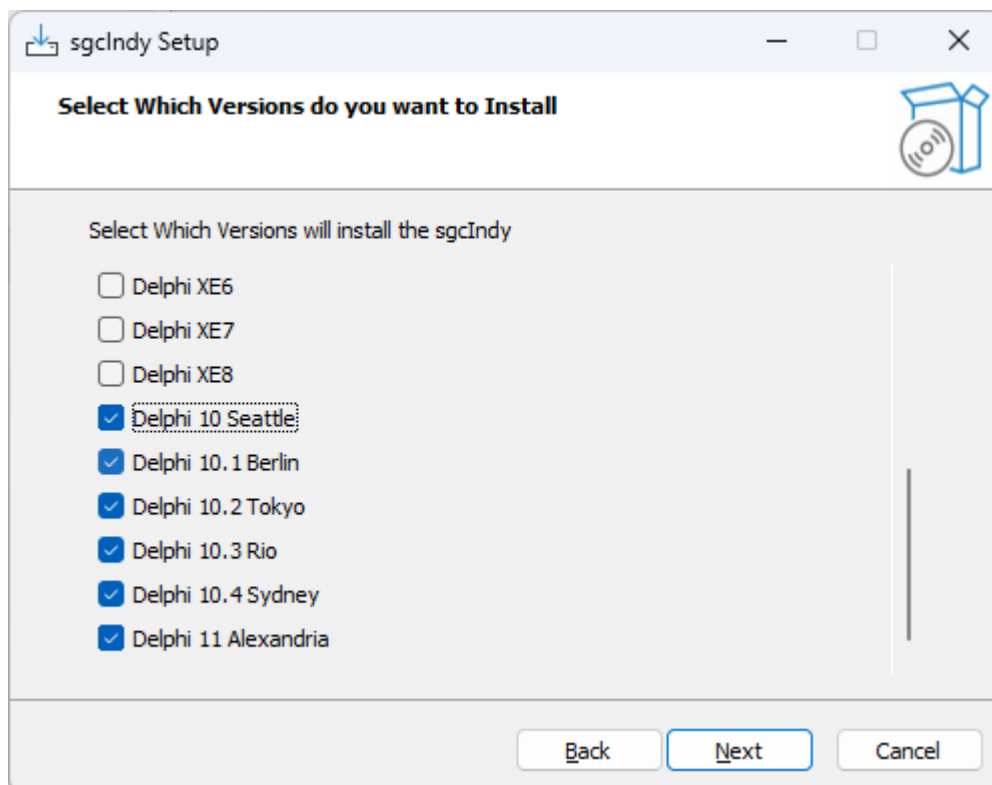
- ユーザーが正常にログインした場合、Delphi、CBuilder、または Rad Studio IDE のどれにインストールするかを選択してください。



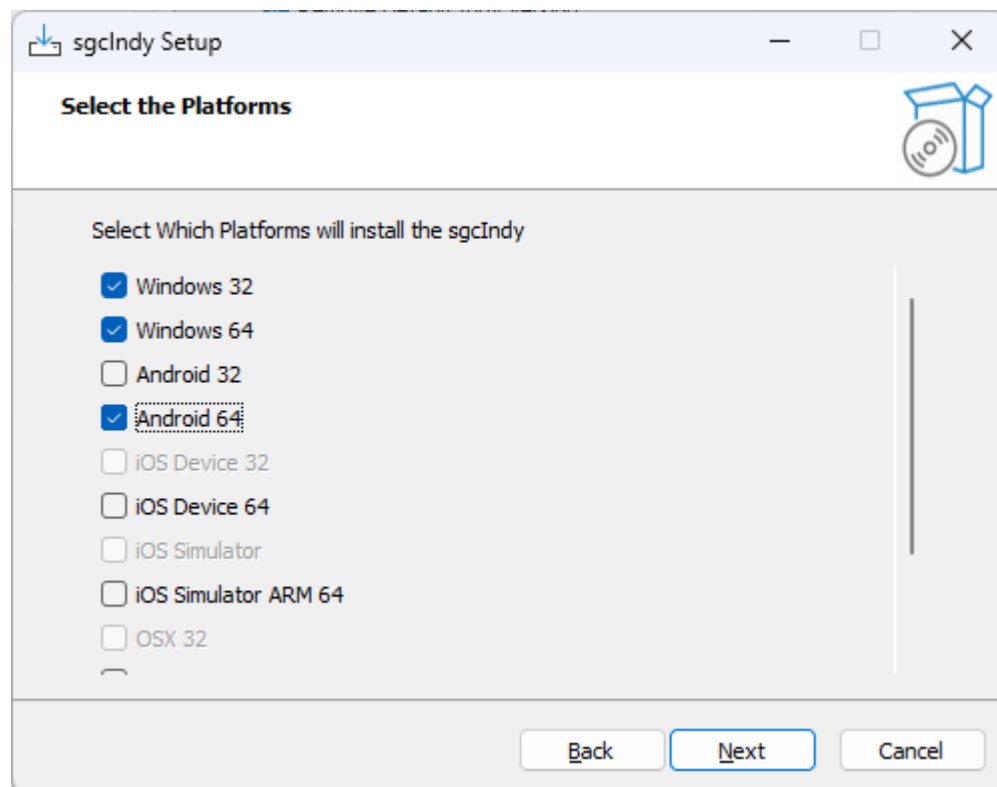
- インストーラーを使用するたびにカスタマイズできるオプションがあります。 **Options** ボタンを押してこれらのプロパティにアクセスします。
 - **Build Packages:** 選択すると、インストーラーはパッケージのビルドを試みます。
 - **Register Paths IDE:** 選択すると、インストーラーはIDEに必要なライブラリパスを登録します。
 - **Register BPLs IDE:** 選択し、インストーラーがパッケージのビルドに成功した場合、インストーラーはデザイン時パッケージをIDEに登録します。
 - **Remove Default Indy Version:** 選択すると、インストーラーはまずRad Studioに付属している標準Indyバージョンをアンインストールします。
 - **Restore Default Indy Version when Uninstalling:** 選択すると、パッケージのアンインストール時にインストーラーはアンインストールされた標準Indyバージョンをロールバックします。
 - **Compatibility Mode:** 選択すると、dcpファイルはバージョンなしでコンパイルされ、Embarcadero/libフォルダーにコピーされます。DevExpressなどの他のパッケージがIndyパッケージを使用している場合はこのオプションを確認します。さらに、Embarcadero IP抽象化ユニットをインストールされているsgclndyバージョンでコンパイルします。
 - **Always use of the following OpenSSL API Versions:** OpenSSL 1.1.1またはOpenSSL 3.0.0 APIの使用を強制したい場合はこのオプションを確認します。



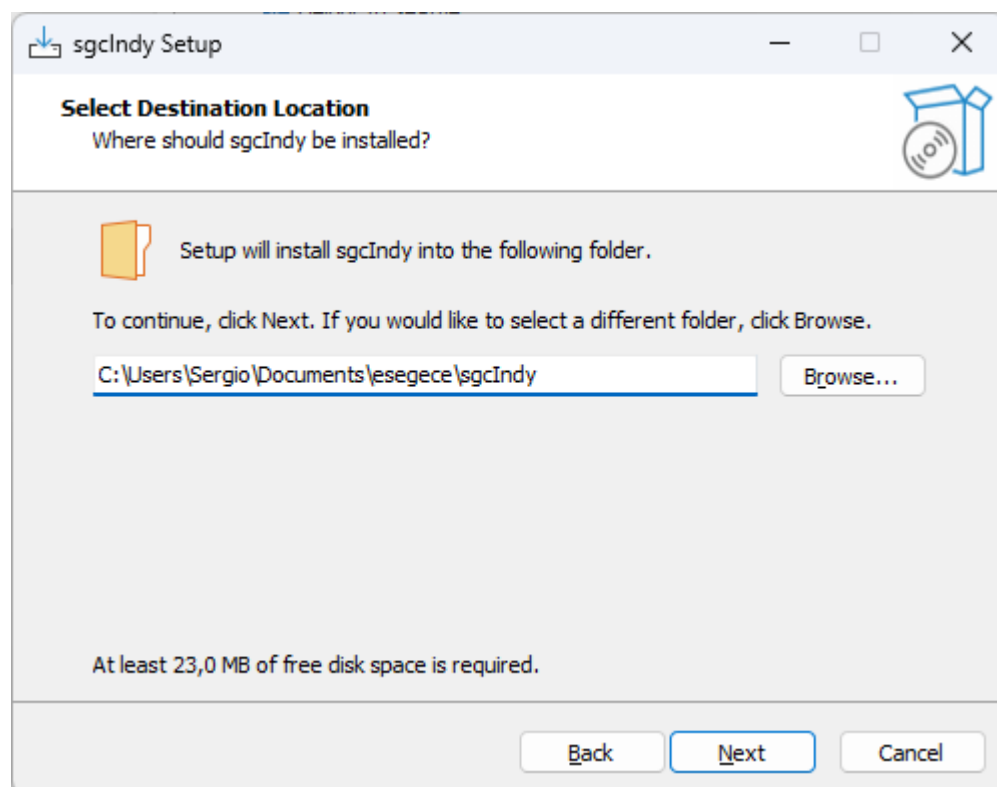
- インストールする IDE バージョンを選択できます。インストーラーがインストール済みとして検出した IDE バージョンのみが選択可能です。



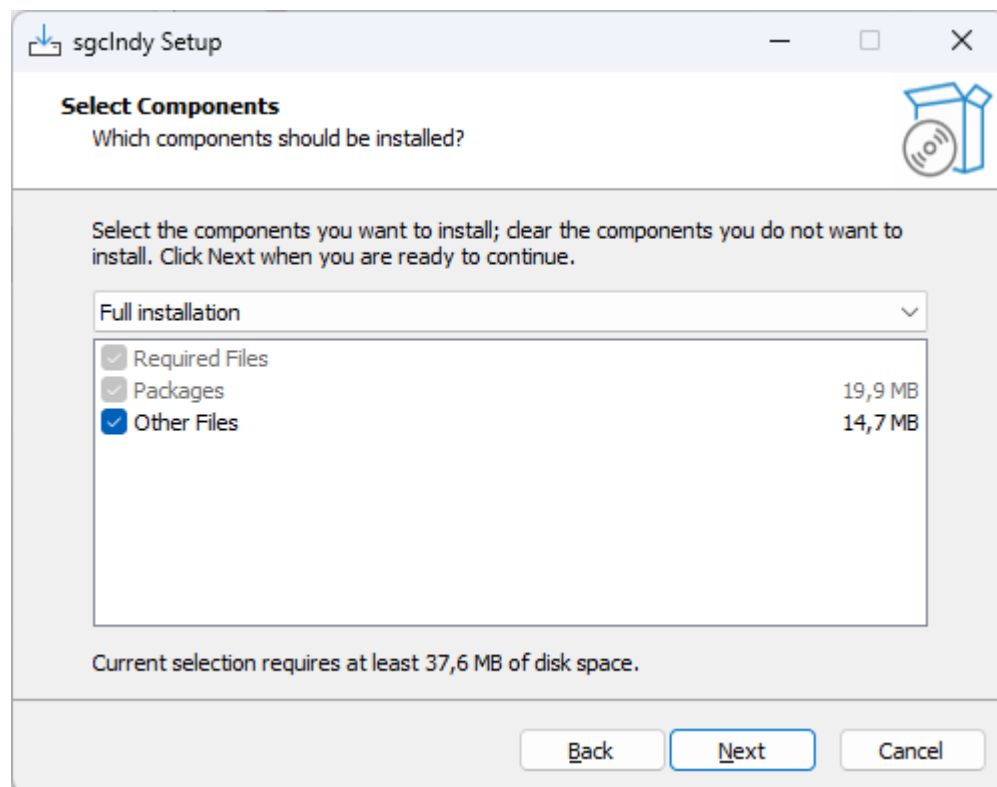
- 次のステップはプラットフォームの選択です。



- パッケージをインストールするフォルダーを選択してください。パッケージを再インストールする場合、インストーラーはデフォルトで前回のインストールで選択したフォルダーを選択します。



- インストールするコンポーネントを選択してください。



- 最後に、ファイルを展開し、パッケージをコンパイルしてインストールし、必要なパスを IDE に登録します。

Install Errors

- MsBuild はライブラリパスが長すぎるとエラーを発生させます。この問題を解決するには、ライブラリパスから未使用のパスを削除してください。MsBuild には32K文字の制限があります。

手動インストール

Indy がすでにインストールされている場合は、まずアンインストールする必要があります。

- プリコンパイルされた **BPL ファイル** (dclIndyCoreX.bpl と dclIndyProtocolsX.bpl) を "コンポーネント > パッケージのインストール" ダイアログから IDE を通じて削除します。
- 次に、**既存のバイナリをすべて削除**します (IndySystemX.、IndyCoreX.、IndyProtocolsX.、dclIndyCoreX.、dclIndyProtocolsX.*)。また、Indy 10 のソース ファイルが存在する場合はそれも削除します。
- IDE の \bin、\lib、\source フォルダー、\Indy サブフォルダー、OS システムフォルダーのファイルを**確認**してください。

sgclndy パッケージをビルドするには、次のいずれかを行います

- (CBuilderのみ)** CBuilderのバージョンに対応するコマンドライン**FULLC#.BAT**スクリプトを使用します。
- 個々の DPK ファイルを IDE で開き、以下の順序でコンパイル**してください:
 - IndySystemX.dpk (Lib\System 内)
 - IndyCoreX.dpk (Lib\Core 内)
 - IndyProtocolsX.dpk (Lib\Protocols 内)
 - dclIndyCoreX.dpk (Lib\Core 内)
 - dclIndyProtocolsX.dpk (Lib\Protocols 内)

Indyパッケージのビルドが完了したら、メニュー**Components / Install Packages**へ移動し、Indy Design-Time Packages をインストールします

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

最後に IDE のパスを sgclIndy パッケージに設定してください。

Configure ZLib

ZLib バージョン: 1.2.12

sgcWebSockets は WebSocket [Compression PerMessage Deflate](#) 拡張が有効な場合に ZLib 圧縮を使用します。デフォルトでは、ZLib はアプリケーションに静的にリンクされているため、ZLib ライブラリを別途デプロイする必要はありません。

特定のライブラリを使用したい場合は、プロジェクトに以下の条件定義を追加してください。

```
SGC_DYNAMICLOAD_ZLIB
```

別の方法として、ソースフォルダーにある `sgcIndy.inc` ファイルを編集して次の行を追加することもできます。

```
{$DEFINE SGC_DYNAMICLOAD_ZLIB}
```

最後に、ZLib ライブラリの場所を設定する必要があります。以下のメソッドを使用して、ライブラリが配置されているフルパス (ライブラリ名なし) を渡してください。

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

**この設定はソースコード付き sgcWebSockets Enterprise Edition にのみ有効です。*

QuickStart

WebSockets コンポーネント

新しいWebSocketサーバーまたはWebSocketクライアントの作成は非常に簡単です。クラスの新しいインスタンスを作成し、Host / Portを設定して、プロパティActive = trueを設定することでプロセスを開始するだけです。

[QuickStart WebSockets](#)

HTTP コンポーネント

HTTP/2プロトコルを使用すると、HTTP/1プロトコルを使用する場合よりもはるかに高速な HTTPサーバー/クライアントを作成できます。HTTP/2サーバーはWebSocketサーバーに含まれており、HTTP/2クライアントはHTTP/2プロトコルを実装する専用のコンポーネントです。

[クイックスタート HTTP](#)

スレッディングフロー

sgcWebSockets コンポーネントはスレッド化されており、**接続はセカンダリスレッドで実行されます**。デフォルトでは、メインイベントはメインスレッドでディスパッチされます。これはディスパッチするイベント数が少ない場合に有用ですが、パフォーマンスを向上させるために、**イベントが接続スレッドのコンテキストでディスパッチされるようにコンポーネントを設定できます**。スレッドフローの設定方法を説明する次の記事を参照してください：

[NotifyEvents の設定方法](#)

アプリケーションのビルド方法

sgcWebSocketsライブラリを使用したアプリケーションのビルドは非常に簡単です。次のヒントに従うだけで、アプリケーションのビルドを成功させることができます。

[Build](#)

高速パフォーマンスサーバー

sgcWebSockets は **2 種類**のサーバー実装を持っています。1つは **Indy** サーバーに基づくもの、もう1つは **HTTP.SYS Microsoft** サーバーに基づくものです。後者は何千もの同時接続を処理する必要がある高パフォーマンスのサーバーに推奨されます。サーバーのパフォーマンスを向上させる方法を説明した以下の記事をご確認ください。

[高性能サーバー](#)

メモリマネージャー

適切なメモリマネージャーを選択すると、アプリケーションのパフォーマンスを向上させることができます。複数のメモリマネージャーを比較した以下の記事をご覧ください。

OpenSSL

アプリケーションがセキュア接続を必要とする場合、通常、通信を暗号化するために**openssl ライブラリ**が必要です。次の手順に従って、openssl ライブラリを使用してアプリケーションを正常に設定してください。

[OpenSSL の設定](#)

Indy

Indy ライブラリは一部の sgcWebSockets コンポーネントの基盤として使用されており、sgcWebSockets Enterprise エディションには openssl 1.1.1 および openssl 3.0.0、ALPN などを使用できるカスタム Indy バージョンが含まれています。

[Indy](#)

Linux (Lazarus)

Lazarus プロジェクトをコンパイルしてこのメッセージが表示された場合：

セマフォの初期化に失敗しました（同時スレッド数が多すぎる可能性があります）。

プロジェクトファイルに **cthreads** ユニットを追加するだけです。

QuickStart | WebSockets

サーバー WebSocket と 2 種類のクライアント WebSocket（アプリケーションクライアントとウェブブラウザクライアント）を作成する基本的な例から始めましょう。

WebSocket Server

1. 新しい Windows Forms アプリケーションを作成します。
2. フォームに TsgcWebSocketServer をドロップします。
3. イベントタブで OnMessage イベントをダブルクリックし、次のコードを入力します:

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

4. Formにボタンを配置し、ダブルクリックして次のコードを入力します:

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

WebSocket クライアント

1. 新しい Windows Forms アプリケーションを作成します。
2. フォームに TsgcWebSocketClient をドロップし、サーバーに接続するために Host と Port プロパティを設定します。
3. フォームに TButton をドロップし、ダブルクリックして次のコードを入力してください:

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

4. Formにボタンを配置し、ダブルクリックして次のコードを入力します:

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

Web ブラウザクライアント

1. 新しいHTMLファイルを作成します
2. テキストエディターでファイルを開き、次のコードをコピーします:

```
<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>
```

このファイル内のホストとポートをカスタムのホストとポートに置き換えてください！

3. ファイルを保存してください。これで基本的な WebSocket ウェブブラウザクライアントの設定が完了しました。

How To Use

1. Serverアプリケーションを起動し、ボタンを押してWebSocket Serverを起動し、新しい接続をリッスンします。
2. クライアントアプリケーションを起動し、button1 を押してサーバーに接続し、button2 を押してメッセージを送信します。サーバー側で、クライアントが送信したテキストのメッセージが表示されます。
3. HTML ファイルを Web ブラウザー（Chrome、Firefox、Safari、または Internet Explorer 10 以降）で開き、「Open」を押して接続を開き、「Send」を押してサーバーにメッセージを送信します。サーバー側では、Web ブラウザークライアントから送信されたテキストのメッセージが表示されます。

Linux コンパイラ

シンプルなサーバーの例（ポート 5000 でリッスン）。

```
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)

Lazarusプロジェクトをコンパイルしてこのメッセージが表示された場合：

セマフォの初期化に失敗しました（同時スレッド数が多すぎる可能性があります）。

cthreads ユニットをプロジェクトファイルに追加してください。

QuickStart | HTTP

HTTP/2 サーバーと HTTP/2 クライアントを作成する必要がある基本的な例から始めましょう。

HTTP/2 サーバー

1. 新しい Windows Forms アプリケーションを作成します。
2. TsgcWebSocketHTTPServer をフォームにドロップします。
3. イベントタブで OnCommandGet イベントをダブルクリックし、以下のコードを入力します:

```
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. デフォルトでは、サーバーはHTTP/1接続のみを有効にするため、プロパティHTTP2Options.Enabled = trueでHTTP/2オプションを有効にし、その後SSL Optionsを設定します。セキュアな接続には[OpenSSLライブラリ](#)が必要です。

```
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. フォームにボタンをドロップし、ダブルクリックして次のコードを入力します:

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

HTTP/1 クライアント

1. 新しい Windows Forms アプリケーションを作成します。
2. フォームに TButton を配置し、ダブルクリックして次のコードを入力します。

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


HTTP/2 クライアント

1. 新しい Windows Forms アプリケーションを作成します。
2. フォームに TButton を配置し、ダブルクリックして次のコードを入力します。

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

クイックスタート | スレッディングフロー

sgcWebSockets コンポーネントはスレッド化されています。例えば、**TsgcWebSocketHTTPServer** (Indy ライブラリベース) は接続ごとに1つのスレッドを作成し、**TsgcWebSocketServer_HTTPAPI** (Microsoft HTTP.SYS ベース) はスレッドプールを実行して接続を処理します (最大64スレッド)。**TsgcWebSocketClient** は独自のスレッドを実行して WebSocket サーバーからのレスポンスを非同期で処理します。

デフォルトでは、**NotifyEvents** というプロパティがあり、値は **neAsynchronous** です。これは、WebSocket クライアントがメッセージを受信すると、そのメッセージはキューに入れられ、後で OS によってメインスレッドでディスパッチされることを意味します。これは、多くのメッセージを受信しないクライアントには適しており、例えばフォーム上のコントロールを更新したい場合にメインスレッドとの同期が不要なため使いやすいです。

しかし、サーバー/クライアントが短時間に多くのメッセージを処理する必要がある場合は、このスレッドフローを接続スレッドのコンテキストでイベントがディスパッチされるものに変更する方が良いです。これを行うには、**NotifyEvents** プロパティを **neNoSync** に設定するだけです。こうすることで、例えばクライアントがサーバーからメッセージを受信した場合、そのメッセージはセカンダリスレッドのコンテキストでディスパッチされます。フォームのコントロールを更新する必要がある場合は、最初にメインスレッドと同期してから、フォームコントロールを更新してください (フォームコントロールはスレッドセーフではないため)。共有オブジェクトにアクセスする場合も同様で、独自の同期メソッドを実装する必要があります。

スレッドフロー 簡易モード (**NotifyEvents = neAsynchronous**) と低パフォーマンス

これはデフォルトのスレッド処理フローであり、通常はデモサンプルで使用されます。1秒あたり複数のメッセージを処理することを想定しておらず、フォームコントロールを更新したり、共有オブジェクトにアクセスしたりする必要がある場合は、このモードを選択してください。

`NotifyEvents = neAsynchronous`

スレッディングフローの最高パフォーマンス (**NotifyEvents = neNoSync**)

高パフォーマンスが必要で、多数のメッセージを処理することが予想されるサーバーコンポーネントおよびクライアントにこのスレッドフローを設定します。この設定を使用すると、イベントは接続スレッドのコンテキストでディスパッチされるため、フォームコントロールを更新するには、最初にメインスレッドと同期する必要があります。

`NotifyEvents = neNoSync`

メインスレッドを同期する方法

`TThread.Synchronize` または `TThread.Queue` を呼び出してメインスレッドと同期できます。両方のメソッドが使用でき、どちらを選択するかは同期の実装方法によって異なります。

`TThread.Synchronize`

このメソッドはブロッキングです。つまり、`Synchronize` を呼び出すと、メインスレッドと同期されるまでコードがブロックされます。

`TThread.Queue`

このメソッドはノンブロッキングです。キューを呼び出すと、メッセージはキューに入れられ、後で送信されます。

使用例 コード

WebSocket クライアントから受信したメッセージでメモを更新します。

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

クイックスタート | ビルド

sgcWebSocketsライブラリでアプリケーションを構築するのは非常に簡単です。コンポーネントがOpenSSLライブラリを必要とするかどうかだけに留意してください。アプリケーションがセキュアな接続を必要とする場合は、OpenSSLライブラリをデプロイする必要があります（クライアントコンポーネントで[Windows用SChannel](#)を使用する場合を除く）。

Windows アプリケーションの場合は、アプリケーションと同じフォルダーに OpenSSL ライブラリを展開するだけで十分です。

その他のパーソナリティについては、以下の記事を確認してください。

- [OSX アプリケーションのビルド](#)
- [Android アプリケーションのビルド](#)
- [iOS アプリケーションのビルド](#)

CBuilder DEBUG

CBuilder を使用して sgcWebSockets ライブラリをデバッグしたい場合は、次の手順に従ってください。

- プロジェクトオプションに移動します。
- Delphi Compiler > Compiling でデバッグ .dcu を使用を有効にします。
- C++ リンカーの「Dynamic RTL とリンク」を無効にします。
- パッケージ > ランタイムパッケージの下で「ランタイムパッケージとのリンク」を無効にします。

VCL/RTL ソースコードにステップインできるようになります。

Build | OSX Application

sgcWebSocketsライブラリを使用してOSXアプリケーションをビルドするには、EmbarcaderoのWebサイトの手順に従ってOSXアプリケーションをビルドする必要があります。

MacOS での PASServer のインストール

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

デベロッパー ID 証明書の取得

有効な Apple Developer Account で <https://developer.apple.com> にログインし、Certificatesメニューから新しい「Developer ID Application」を作成します。

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

新しい Apple ID を作成する

次に <https://appleid.apple.com/account> にアクセスして新しい Apple ID を作成します。

Configure Provisioning

最後に、メニュー **Project / Options / Provisioning** を開き、OSXアプリケーションを公証するために必要なデータを入力します。

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

プロジェクトにライブラリが必要な場合は、メニューの **Project / Deployment** に含めることを忘れないでください。Remote Path を「Contents\MacOS\」に設定してください。

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

これらのライブラリは、アプリケーションがノータライズされる際に自動的に署名されます。ライブラリが署名されているかどうかは、以下のコマンドを使用して確認できます：

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

[openssl OSXの構成方法](#)について詳しくお読みください。

Build | Android アプリケーション

sgcWebSockets ライブラリを使用して Android アプリケーションを構築するには、Embarcadero の Web サイトの手順に従って Android アプリケーションを構築する必要があります。

Android アプリの作成

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

プロジェクトデプロイメント

プロジェクトが何らかのライブラリを必要とする場合は、メニュー **Project / Deployment** に含めることを忘れないでください。Remote Path を「`.\assets\internal`」に設定します。

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の構成 方法について詳しくお読みください。

Build | iOS Application

sgcWebSockets ライブラリを使用して iOS アプリケーションをビルドするには、iOS アプリケーションのビルドについて Embarcadero ウェブサイトの手順に従う必要があります。

MacOS での PASServer のインストール

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

iOS 開発証明書の取得

有効な Apple Developer Account で <https://developer.apple.com> にログインし、Certificates メニューから新しい「iOS Development Certificate」を作成してください。
iOS アプリ用の新しい識別子と新しいプロビジョニングプロファイルを作成します。

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

Bundle Identifier の設定

メニュー **Project / Options / Application / Version Info** を開き、バンドル識別子を設定します

Version Info

Target
Release configuration - iOS Device 64-bit platform

☒ Include version information in project

Module version number

Major version	Minor version	Build
1	0	0

Build number options
Do not change build number

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

Deployment

プロジェクトで一部の静的ライブラリが必要な場合は、Embarcadero lib/iosDevice64 フォルダーにこれらのライブラリをコピーしてください：

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

[iOS の OpenSSL 設定方法](#)の詳細をご覧ください。

プロビジョニング

最後に、メニューの **Project / Options / Deployment** で証明書が正常に読み込まれているか確認してください。

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

高速パフォーマンスサーバー

Indy ライブラリベースのサーバー

`TsgcWebsocketServer` と `TsgcWebsocketHTTPServer` は Indy ライブラリに基づいているため、すべての接続はスレッドで処理されます。1000 の同時接続がある場合、これらの接続を処理するために少なくとも 1000 のスレッドが必要です。パフォーマンスが重要な場合は、パフォーマンスを向上させサーバーの動作を改善するための「調整」を行う必要があります。**sgcWebSockets 4.3.3 以降、Indy サーバーは IOCP もサポートしています。詳細を読む。**

サーバーのパフォーマンスを向上させるには、次のヒントを使用してください。

1. サーバーコンポーネントのプロパティ **NotifyEvents := neNoSync** を設定します。これは、イベントが接続スレッドのコンテキストで発生し、同期メカニズムがないことを意味します。VCL コントロールや共有オブジェクトにアクセスする必要がある場合は、独自の同期メカニズムを使用してください。

2. サーバーコンポーネントのプロパティ **Optimizations.Connections.Enabled := True** を設定します。サーバーで 1000 以上の同時接続を持ち、`Server.WriteData` メソッドを頻繁に呼び出す場合は、このプロパティを有効にしてください。基本的に、Indy 接続リストへのアクセスよりも高速な検索が可能なキャッシュリストに接続を保存します。

2.1 `CacheSize`: 高速キャッシュに保存される接続数です。デフォルト値 = 100。

2.2 `GroupLevel`: 最初の文字でいくつかのリストを内部的に作成します。接続が多い場合、検索が高速になります。デフォルト値 = 1。

3. サーバーコンポーネントのプロパティ **Optimizations.Channels.Enabled := True** を設定します。このプロパティを有効にすると、チャンネルがリストに保存され、以前の方法よりも高速に検索できます。

4. サーバーコンポーネントのプロパティ **Optimizations.ConnectionsFree.Enabled := True** を設定します。このプロパティを有効にすると、切断が発生するたびに `TsgcWSCConnection` を破棄する代わりに、オブジェクトをリストに保存し、X 秒ごとにリストに保存されたすべてのオブジェクトを破棄します。このプロパティを有効にするとメモリ消費量が増加するため、短時間に多くの切断がある場合は `false` に設定してください。

4.1 `Interval`: 切断されたすべての接続がリストに格納されて破棄されるまでの秒数。デフォルトは 60 です。

5. デフォルトでは、`sgcWebSockets` は共有オブジェクトへのアクセスを保護するために **クリティカルセクション** を使用します。ただし、クリティカルセクションの代わりに `TMonitor` または `SpinLock` を使用することもできます。以下のコンパイラ定義のいずれかを指定してプロジェクトをコンパイルするだけです

3.1 `{ $DEFINE SGC_SPINLOCKS }`

3.2 `{ $DEFINE SGC_TMONITOR }`

6. 最新の **FastMM4** を使用してください。 <https://github.com/pleriche/FastMM4> からダウンロードできます。FastMM4 は非常に優れたメモリマネージャーですが、マルチスレッドアプリケーションではスケールしないことがあります。アプリケーションで以下のコンパイラ定義を使用してください：

`{ $DEFINE UseReleaseStack }`

次に、プロジェクトの `uses` の最初のユニットとして **FastMM4** を追加して再コンパイルしてください。高並行サーバーでは、パフォーマンスの向上が見られます。

この調整は以下のことを行います：FreeMem 呼び出し中にブロックを直ちに解放できない場合、ブロックはバックグラウンドクリーンアップスレッドで、または次の FreeMem 呼び出し中に後で解放されるブロックのリストに追加されます。

7. FastMM4 より優れた最新の **FastMM5** をダウンロードできます： <https://github.com/pleriche/FastMM5>

これは FastMM4 と同じ開発者による新しいバージョンで、Delphi XE3 以降をサポートし、Windows32 および Windows64 で使用できます。

FastMM5 はデュアルライセンスで、GPL とコマーシャルの2つのライセンスがあります。商業プロジェクトで使用する場合は、ライセンスを購入する必要があります。

以下に、FastMM4 と FastMM5 のパフォーマンス比較グリッドを示します。1、10、100、500、1000 の同時クライアントを使用して 100,000 回の WebSocket リクエストとレスポンスを実行しています。マルチスレッドアプリケーションでは FastMM5 の方が FastMM4 よりもパフォーマンスがはるかに優れています。

クライアント	Windows	FMM4	FMM5	差分
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 サーバー Windows

sgcWebSockets Enterprise EditionはWindowsで**IOCP**をサポートしています。つまり、接続ごとに1スレッドを作成する代わりに、スレッドプールがすべての接続を処理します。IOCPを有効にするには、IOHandlerをIOCPに設定します。

```
IOHandlerOptions.IOHandlerType = iohIOCP
```

プロパティ IOHandlerOptions.IOCP を使用して IOCP プロパティをカスタマイズできます。

- **IOCPThreads:** 接続の処理に使用されるスレッドです。デフォルト値は0で、プロセッサ数を使用してスレッドが自動的に計算されることを意味します（Delphi 7からDelphi 2007では、CPU数取得関数がサポートされていないためこの値は32に設定されます）。
- **WorkOpThreads:** 各接続のリクエストを常に同じスレッドで処理させたい場合は、ゼロより大きい値を設定します。デフォルトでは、IOCPリクエストはランダムなスレッドで処理されます。接続を常に同じスレッドで処理させたい場合は、ゼロより大きい値を設定してください。例: WorkOpThreads = 32を設定すると、サーバーは32個のスレッドを作成し、新しいリクエストがあるたびに、その接続が以前に処理済みであれば同じスレッドにキューイングされます。

数千の同時接続を処理したい場合は IOCP を推奨します。

Indy サーバー Linux

sgcWebSockets Enterprise Edition は Linux 上で **EPOLL** をサポートしています。これにより、接続ごとに1スレッドを作成するのではなく、スレッドプールがすべての接続を処理します。EPOLL を有効にするには、IOHandlerをEPOLLに設定するだけです。

IOHandlerOptions.IOHandlerType = iohEPOLL

IOHandlerOptions.EPOLLプロパティを使用すると、EPOLLプロパティをカスタマイズできます。

- **EPOLLThreads:** 接続を処理するために使用されるスレッドです。デフォルト値はゼロで、プロセッサ数を使用してスレッドが自動的に計算されます。
- **WorkOpThreads:** 各接続のリクエストを常に同じスレッドで処理したい場合は、ゼロより大きい値を設定します。デフォルトでは、EPOLL リクエストはランダムなスレッドで処理されます。接続が常に同じスレッドで処理されるようにしたい場合は、ゼロより大きい値を設定してください。例: WorkOpThreads = 32 に設定すると、サーバーは 32 スレッドを作成し、新しいリクエストが来るたびに、その接続が以前に処理済みであれば同じスレッドのキューに入れられます。

EPOLL は何千もの同時接続を処理する場合に推奨されます。

HTTP.SYS ベースのサーバー

[TsgcWebSocketServer_HTTPAPI](#) コンポーネントは Microsoft HTTP API に基づいており、IOCP で動作するように設計されています。そのため、サーバーが何千もの接続を処理する必要がある場合に推奨されますが、Windows でのみ実行できるという制限があります。

サーバーは同じポートで**WebSocket**および**HTTP/2**プロトコルを処理でき、一部のエンドポイントのみを処理するように設定できるため、他の実装と連携できます。

例：このサーバーを `sgcWebSockets` ライブラリで `WebSocket` 接続を処理するように設定し、他の実装/サードパーティなどが他のエンドポイントを使用できるようにします。

- エンドポイント: `https://server/ws` は `sgcWebSockets` を使用した `WebSocket` プロトコルを使用する接続を処理します
- エンドポイント: `https://server/other` は別のライブラリを使用して接続を処理します。

最新の**FastMM5**を使用してください。ダウンロードは次から行えます: <https://github.com/plieriche/FastMM5>
これはFastMM4と同じ開発者による新しいバージョンで、Delphi XE3以降をサポートし、Windows32およびWindows64で使用できます。

FastMM5 はデュアルライセンスで、GPL とコマーシャルの2つのライセンスがあります。商業プロジェクトで使用する場合は、ライセンスを購入する必要があります。

以下に、FastMM4 と FastMM5 のパフォーマンス比較グリッドを示します。1、10、100、500、1000 の同時クライアントを使用して 100,000 回の `WebSocket` リクエストとレスポンスを実行しています。マルチスレッドアプリケーションでは FastMM5 の方が FastMM4 よりもパフォーマンスがはるかに優れています。

クライアント	Windows	FMM4	FMM5	差分
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%

メモリアネージャー

最近、Pierre le Riche によって開発された FastMM の新しいバージョンがリリースされました。新しいバージョンは **FastMM5** と呼ばれ、マルチスレッドアプリケーションのパフォーマンスを向上させるために書き直されており、速度優先またはメモリ使用量削減のために設定可能で、その他の改善も含まれています。

Delphi **XE3** コンパイラ以降をサポートし、**Windows32** および **Windows64** で使用できます。

FastMM5 はデュアルライセンスであり、**GPL** と **商用** の 2 種類のライセンスがあります。商用プロジェクトで使用する場合はライセンスを購入する必要があります。詳細はこちら

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

FastMM4 には、Maxim Masiutin によって開発された **FastMM4-AVX** という新しいフォークがあり、より効率的な同期、より高速なメモリコピーのための AVX 命令、速度の向上などの非常に興味深い機能を追加しています。FastMM4-AVX はデュアルライセンスです: MPL と GPL。詳細はこちら:

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

設定

当社のコンポーネントでパフォーマンスをテストするために、新しい Windows コンソール アプリケーション **sgcBenchmark.exe** が作成されました。これは、sgcWebSockets コンポーネントを使用して各メモリアネージャーのパフォーマンスを測定するために使用されます。

テストは非常にシンプルです。クライアント（1台以上）がサーバーに接続し、サーバーにメッセージを送信すると、サーバーは同じメッセージをクライアントに返します。これを 100,000 回繰り返します。テストは同時クライアント数を変えて繰り返されます。最初は 1 台、次に 10 台、100 台という順で行います。測定時間は、クライアントが最初のメッセージを送信してからサーバーから最後のメッセージを受信するまでの経過時間です（サーバーへの接続に使用される時間は測定されません）。

このベンチマークでは、Delphi 10.4.1 に付属するデフォルトのメモリアネージャー、FastMM5、FastMM4-AVX を使用したパフォーマンスを比較します。

Benchmark Indy WebSocket サーバー

最初のベンチマークでは、使用するサーバーは **Indy WebSocketサーバー** です。このサーバーは Indy TCP サーバーをベースとしているため、接続ごとに 1 つのスレッドが作成されます。

値はミリ秒単位で計測されています。例えば、Windows32 プラットフォームで 1 クライアントによる最初のテストでは、デフォルトのメモリアネージャーを使用すると 4135 ミリ秒、FastMM5 では 4214 ミリ秒、FastMM4-AVX では 4823 ミリ秒かかります。計算されたパーセンテージは基準値（この場合は Delphi 付属のデフォルトのメモリアネージャー）に対するものです。パーセンテージが低いほどパフォーマンスが優れています。

ベンチマークは 3 回実施され、表示される値はベンチマークの合計 / 3 です。

ベンチマークで使用されたサーバーは次のとおりです:

- Windows 2016 Server Datacenter
- 16 仮想プロセッサ
- 32 GB RAM
- 2.2 GHz

使用された Delphi バージョンは Delphi 10.4.1 で、最新の FastMM5 と FastMM4-AVX バージョンは GitHub サーバーからダウンロードしました。

以下にベンチマーク結果を示します。

クライアント	プラットフォーム	デフォルト (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%

HTTP.SYS サーバーのベンチマーク

2 つ目のベンチマークで使用するサーバーは、[HTTP.SYS WebSocket サーバー](#)です。このサーバーは HTTP API Microsoft Framework をベースにしており、接続はスレッドプールで処理されます。

値はミリ秒単位で測定されます。例えば、Windows 32 プラットフォームで 1 クライアントを使用した最初のテストでは、デフォルトのメモリマネージャーで 5364 ミリ秒、FastMM5 で 5182 ミリ秒、FastMM4-AVX で 5838 ミリ秒かかります。計算されたパーセンテージは参照値（この場合は Delphi に付属のデフォルトのメモリマネージャー）に対して計算され、パーセンテージが低いほどパフォーマンスが優れています。

ベンチマークは 3 回実施され、表示される値はベンチマークの合計 / 3 です。

ベンチマークで使用されたサーバーは次のとおりです：

- Windows 2016 Server Datacenter
- 16 仮想プロセッサ
- 32 GB RAM
- 2.2 GHz

使用された Delphi バージョンは Delphi 10.4.1 で、最新の FastMM5 と FastMM4-AVX バージョンは GitHub サーバーからダウンロードしました。

以下にベンチマーク結果を示します。

クライアント	プラットフォーム	デフォルト (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%

ベンチマークに関するコメント

3 種類のメモリマネージャーのベンチマーク後に得られた結果についていくつかのコメントを以下に示します:

- シングルスレッドアプリケーションで使用する場合、FastMM4、FastMM5、 および FastMM4-AVX の間でパフォーマンスに大きな差はありません。
- **FastMM5** と **FastMM4-AVX** は、マルチスレッドアプリケーションではるかに優れた動作をします。
- FastMM5 と FastMM4-AVX の差異は、少なくともこれらのベンチマークでは小さいです。
- **Windows 32** ベンチマークは、**Windows 64** テストよりも優れたパフォーマンスを発揮します。 Windows 64 アプリケーションで FastMM5 または FastMM4-AVX を使用すると、 Windows 32 よりもパフォーマンスが向上します。

どのメモリーマネージャーを選択するか最終的な判断はプロジェクトによって異なります。すべての条件で最善となる単一のメモリーマネージャーは存在しないと思われるので、どちらかを選択する前に、ニーズに合ったパフォーマンスを発揮するものを何度も繰り返しテストしてください。

OpenSSL

OpenSSL は、コンピューターネットワーク上の通信を盗聴から守ったり、相手側を識別したりする必要があるアプリケーション向けのソフトウェアライブラリです。HTTPS ウェブサイトの大多数を含む多くのインターネットサーバーで広く使用されています。

このライブラリは、セキュア接続が必要な場合にIndy Libraryに基づくコンポーネントによって必要とされます。アプリケーションがOpenSSLを必要とする場合は、アプリケーションをデプロイする前にファイルシステムに必要なファイルを用意する必要があります:

現在、sgcWebSocketsは次のopensslバージョンをサポートしています: **1.0.2**、**1.1**、および**3.0** から **3.3**。

プラットフォーム	API 1.0	API 1.1	API 3.*	静的/動的リンク
Windows (32ビットおよび64ビット)	libeay32.dll および ssleay32.dll	libcrypto-1_1.dll and libssl-1_1.dll	libcrypto-3.dll and libssl-3.dll	ダイナミック
OSX	libcrypto.dylib, libssl.dylib	libcrypto.1.1.dylib, libssl.1.1.dylib	libcrypto.3.dylib, libssl.3.dylib	ダイナミック
iOS Device (32-bit and 64-bit)	libcrypto.a と libssl.a	libcrypto.a と libssl.a	libcrypto.a と libssl.a	Static
iOS Simulator	libcrypto.dylib, libssl.dylib	libcrypto.1.1.dylib, libssl.1.1.dylib	libcrypto.3.dylib, libssl.3.dylib	ダイナミック
Androidデバイス	libcrypto.so, libssl.so	libcrypto.so, libssl.so	libcrypto.so, libssl.so	ダイナミック

各プラットフォームの **OpenSSL** ライブラリの設定方法を以下に示します。

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

OpenSSL の設定

sgcWebSockets の Indy ベースのコンポーネントでは、いくつかの OpenSSL プロパティを設定できます。次のプロパティにアクセスしてください。

- サーバーコンポーネント: `SSLOptions.OpenSSL_Options`.
- クライアントコンポーネント: `TLSOptions.OpenSSL_Options`.

API バージョン

Standard Indy ライブラリは **1.0.2 OpenSSL** ライブラリのみをロードを許可します。これらのライブラリは非推奨となり、最新の OpenSSL リリースは 1.1.1 API を使用しています。

sgcWebSockets Enterprise では、**1.1.1 openssl** ライブラリを読み込むことができます。このプロパティで、どの openssl API バージョンを読み込むかを構成できます。プロセスごとに読み込める API バージョンは1つだけです (そのため、同じアプリケーションで openssl 1.0.2 と 1.1.1 のライブラリを混在させることはできません)。

LibPath

このプロパティを使用して OpenSSL ライブラリの場所を設定できます。Android または OSX プロジェクトで OpenSSL ライブラリの場所を設定する必要がある場合に役立ちます。

以下の値を受け付けます:

- **oslpNone:** この値はライブラリパスの値を設定しません（デフォルト値です）。
- **oslpDefaultFolder:** このフォルダーは OpenSSL ライブラリのデフォルトフォルダーを設定します。このパスはパーソナリティ（Windows、OSX など）ごとに異なります。

追加のOpenSSL関数の読み込み

デフォルトで定義されていない追加の OpenSSL 関数をロードするためにコールバックを使用します。詳細は [OpenSSL 追加関数のロード](#) をご覧ください。

暗号

サーバーで TLS 1.2 および 1.3 のサポートを提供し、最高のセキュリティとパフォーマンスを使用する場合は、次の設定を使用します。

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

次の暗号スイートリストを設定します。

```
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
```

自己署名証明書

テスト目的で自己署名証明書を使用できます。自己署名証明書を作成するには以下のコマンドを実行するだけです。

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

2 つのファイルが作成されます: cert.pem (証明書) と key.pem (秘密鍵)。両方のファイルを 1 つのファイルに結合できます。新しいファイルを作成し、両方のファイルの内容をそこにコピーします。構造は次のようになります。

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

一般的なエラー

SSL_GET_RECORD: wrong version number

このエラーは、サーバーとクライアントが異なるバージョンのSSL/TLSプロトコルを使用していることを意味します。これを修正するには、サーバーおよび/またはクライアントコンポーネントで正しいバージョンを設定してみてください。

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

SSL3_GET_RECORD: decryption failed or bad record mac

通常このエラーは次の場合に発生します:

1. 最新の OpenSSL バージョンを使用しているか確認してください。古すぎる場合は最新のサポートバージョンにアップデートしてください。
2. このエラーがランダムに発生する場合、通常は複数のスレッドが OpenSSL 接続に同時にアクセスしているためです。NotifyEvents = neNoSync を設定してみてください。これは OnConnect、OnDisconnect、OnMessage などのイベントが接続スレッドのコンテキストで発火することを意味し、同期の問題を回避してパフォーマンスを向上させます。欠点として、たとえばメッセージ受信時にフォームのビジュアルコントロールを更新する場合は、ビジュアルコントロールはスレッドセーフではないため、独自の同期メソッドを実装する必要があります。

OpenSSL | Windows

32ビット版と64ビット版があります。アプリケーションと同じフォルダーか、システムパスにこれらのライブラリをコピーする必要があります。

オペレーティングシステムが Windows 32 ビットの場合は、System32 フォルダーにコピーするだけです。

OS が Windows 64 ビットの場合は、64 ビット版を System32 フォルダーに、32 ビット版を SysWow64 フォルダーにコピーしてください。

API 1.0

以下のライブラリが必要です。

- libeay32.dll
- ssleay32.dll

オペレーティングシステムが Windows 32 ビットの場合は、System32 フォルダーにコピーするだけです。

OS が Windows 64 ビットの場合は、64 ビット版を System32 フォルダーに、32 ビット版を SysWow64 フォルダーにコピーしてください。

最新のライブラリはアカウントからダウンロードできます（ライブラリには外部依存関係がなく、デジタル署名されています）。

API 1.1

以下のライブラリが必要です。

Windows 32

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

Windows 64

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

オペレーティングシステムが Windows 32 ビットの場合は、System32 フォルダーにコピーするだけです。

OS が Windows 64 ビットの場合は、64 ビット版を System32 フォルダーに、32 ビット版を SysWow64 フォルダーにコピーしてください。

最新のライブラリはアカウントからダウンロードできます（ライブラリには外部依存関係がなく、デジタル署名されています）。

API 3.*

以下のライブラリが必要です。

Windows 32

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

Windows 64

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

オペレーティングシステムが Windows 32 ビットの場合は、System32 フォルダーにコピーするだけです。OS が Windows 64 ビットの場合は、64 ビット版を System32 フォルダーに、32 ビット版を SysWow64 フォルダーにコピーしてください。

最新のライブラリはアカウントからダウンロードできます（ライブラリには外部依存関係がなく、デジタル署名されています）。

p12証明書を使用している場合は、legacy.dll ライブラリのデプロイが必要です。詳しくは[OpenSSL p12 証明書](#)をご覧ください。

OpenSSL | OSX

新しいバージョンの OSX には openssl ライブラリが含まれていないか、古すぎるため、アプリケーションと一緒にデプロイする必要があります。次の手順でこれらのライブラリをデプロイしてください：

- プロジェクトでProject/Deploymentを開きます。
- 必要なライブラリを追加します。
- RemotePath = 'Contents\Macos\' に設定します。
- OpenSSL の LibPath をデフォルトフォルダーに設定します：
 - Client.TLSOptions.OpenSSL_Options.LibPath = oslpDefaultFolder.
 - Server.SSLOptions.OpenSSL_Options.LibPath = oslpDefaultFolder.

API 1.0

以下のライブラリが必要です。

- libcrypto.dylib
- libssl.dylib

アカウントから最新のライブラリをダウンロードできます。

API 1.1

以下のライブラリが必要です。

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

32 ビット版と 64 ビット版の 2 種類があります。アプリケーションが配置されているフォルダーにこれらのライブラリをコピーする必要があります。

アカウントから最新のライブラリをダウンロードできます。

API 3.0

以下のライブラリが必要です。

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

64ビット版のみ提供されています。これらのライブラリはアプリケーションと同じフォルダにコピーする必要があります。

アカウントから最新のライブラリをダウンロードできます。

OSX アプリケーションに OpenSSL ライブラリを含める場合、アプリケーションが公証（Notarize）された後、ライブラリは署名されます。次のコマンドで確認できます：

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

以下の動画で、MacOSX64 アプリケーションを OpenSSL ライブラリでビルドする方法を確認してください。

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

エラー

クライアントはバージョンなしの `libcrypto.dylib` をロードすべきではありません。安定した ABI がないためです。

MacOS Monterey 以降では、openssl ライブラリを読み込もうとするとこのエラーが発生することがあります。バージョンなしの openssl ライブラリ（例：libcrypto.dylib）を最初に読み込もうとしたときに発生します。このエラーを修正するには、プロパティ `OpenSSL.Options.UnixSymLinks` に値 `oslsSymLinksDontLoad` を設定します。これにより、バージョンなしの OpenSSL ライブラリの読み込みを回避できます。

OpenSSL | Android

新しいバージョンの Android には OpenSSL ライブラリが含まれていないか、古すぎるため、アプリケーションと一緒にデプロイする必要があります。次の手順を使用してこれらのライブラリをデプロイしてください:

- プロジェクトでProject/Deploymentを開きます。
- 必要なライブラリを追加します。
- RemotePath = '.\assets\internal' を設定します。
- OpenSSL の LibPath をデフォルトフォルダーに設定します:
 - Client.TLSOptions.OpenSSL_Options.LibPath = oslpDefaultFolder.
 - Server.SSLOptions.OpenSSL_Options.LibPath = oslpDefaultFolder.

API 1.0

以下のライブラリが必要です。

- libcrypto.so
- libssl.so

アカウントから最新のライブラリをダウンロードできます。

Android 64 ビットでは、TLS 1.2 を使用すると次のエラーが発生する場合があります:

INT_RSA_VERIFY:bad signature

これはAPI 1.1で修正されたOpenSSLエラーです。

(このエラーを回避するためにサーバーがまだこれらの暗号化方式をサポートしている場合は) TLS 1.0または TLS 1.1の使用を試みることができます。

API 1.1

以下のライブラリが必要です。

- libcrypto.so
- libssl.so

アカウントから最新のライブラリをダウンロードできます。

API 3.0

以下のライブラリが必要です。

- libcrypto.so
- libssl.so

アカウントから最新のライブラリをダウンロードできます。

OpenSSL | iOS

64ビットのiOSデバイスにOpenSSLをインストールするには、libcrypto.a と libssl.a のSSLライブラリファイルをシステムにコピーする必要があります。iOS OpenSSLの .zip をダウンロードし、解凍して lib ディレクトリ内の .a ファイルを見つけます。libcrypto.a と libssl.a のSSLライブラリファイルを以下のディレクトリにコピーする必要があります:

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

sgcldSSLOpenSSLHeaders_static (sgcWebSockets のエディションが Enterprise でない場合は IdSSLOpenSSLHeaders_static) ユニットをuses句に追加します。

任意のファイルをデプロイする必要がある場合は、RemotePath = StartUp\Documents に設定し、ファイルを読み込むには以下を使用します (uses 句に System.IOUtils を追加する必要があります) :

TPath.GetDocumentsPath + PathDelim + <your filename>

opensslライブラリは、iOSではメニューProject/Deploymentを使用してデプロイしてはなりません。

API 1.1

IdCompilerDefines.inc を変更し、IOSセクションでSGC_OPENSSL_API_1_1を有効にします:

```
{ $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
// プロジェクトに静的 OpenSSL ライブラリを追加し、以下を include する
// IdSSLOpenSSLHeaders_static unit in their uses clause. It hooks up the
// statically linked functions for the IdSSLOpenSSLHeaders unit to use...
{ $DEFINE STATICLOAD_OPENSSL }
// sgc--> openssl API 1.1向けに有効化
{ $DEFINE SGC_OPENSSL_API_1_1 }
{ $ENDIF }
{ $ENDIF }
```

アカウントからライブラリをダウンロードできます。

API 3.0

IdCompilerDefines.inc を変更して、IOS セクションで SGC_OPENSSL_API_1_1 と SGC_OPENSSL_API_3_0 を有効にしてください:

```
{ $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
// プロジェクトに静的 OpenSSL ライブラリを追加し、以下を include する
// IdSSLOpenSSLHeaders_static unit in their uses clause. It hooks up the
// statically linked functions for the IdSSLOpenSSLHeaders unit to use...
```

```
{ $DEFINE STATICLOAD_OPENSSL}  
// sgc--> openssl API 1.1向けに有効化  
{ $DEFINE SGC_OPENSSL_API_1_1}  
// sgc--> enable for openssl API 3.0  
{ $DEFINE SGC_OPENSSL_API_3_0}  
{ $ENDIF}  
{ $ENDIF}
```

アカウントからライブラリをダウンロードできます。

OpenSSL | 独自の CA 証明書

Github 投稿

CA によって署名され、CA 証明書をローカルマシンに追加した後にウェブブラウザ（Chrome など）が信頼できる証明書を作成するには。

1. プロンプトなしで証明書を作成するための設定ファイルを準備します

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 秘密鍵と証明書を生成する（5年間有効）

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

3. Web サーバーの秘密鍵と CSR を生成します。

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

4. 証明書を作成し、独自の証明機関で署名します（有効期間 1 年）。

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

5. 出力ファイルは以下のとおりです:

- CA.cnf → OpenSSL CA 設定ファイル。証明書作成プロセス後に削除できます。
- CA_cert.pem → [証明機関] 証明書。この証明書をブラウザのローカル認証局ストレージに追加して、この CA で作成されたすべての証明書を信頼させる必要があります。
- CA_cert.srl → ランダムなシリアル番号。証明書作成プロセス後に削除できます。
- CA_key.pem → 新しい [localhost] 証明書を作成する際に使用する必要があります。証明書の作成プロセス後（CA_cert.pem を再利用する予定がない場合）は削除しても構いません。
- localhost.cnf → OpenSSL SSL証明書設定ファイル。証明書作成プロセス後に削除できます。
- localhost.csr → 証明書署名リクエスト。証明書作成プロセス後に削除できます。
- localhost_cert.pem → SSL 証明書。サーバーの `SSLOptions.CertFile` プロパティで設定する必要があります。
- localhost_key.pem → シークレットキー。サーバーの `SSLOptions.KeyFile` プロパティにインストールする必要があります。

OpenSSL | P12 証明書

OpenSSL 3.0 は、廃止または安全でないいくつかのアルゴリズムを legacy provider と呼ばれる内部ライブラリモジュールに移動しました。デフォルトでは読み込まれないため、OpenSSL を暗号化操作に使用するアプリ（またはその言語ランタイム）は、証明書の読み込みやメッセージダイジェストの作成などの際にそれらのアルゴリズムを使用できません。

レガシープロバイダーのアルゴリズムには、MD2、MD4、MDC2、RMD160、CAST5、BF (Blowfish)、IDEA、SEED、RC2、RC4、RC5、DES (ただし 3DES を除く) が含まれます。

セキュリティ上の理由から、これらのレガシーアルゴリズムの使用を廃止することを強くお勧めします。

アプリケーションが、RC2-40-CBC などのレガシー暗号で暗号化されたファイルに保存されたクライアント証明書を使用している場合、openssl プログラムで再暗号化することにより、証明書ファイルを「最新化」できます。

例えば、ローカルコンピューターに client.p12 (または client.pfx) 証明書ファイルがある場合。

```
$ openssl pkcs12 -legacy -in client.p12 -nodes -out cert-decrypted.tmp
(パスワードを求められた場合は入力してください)
```

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

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

エクスポートされた client-new.p12 証明書ファイルには同じキーが含まれていますが、AES-256-CBC を使用して暗号化されています。

以下のsgcWebSocketsおよびsgcIndyパッケージの構成を確認してください:

sgcWebSockets

- プロパティ **OpenSSL_Options.Legacy.Enabled** を True に設定します。
- レガシーライブラリの場所を設定してください。
 - **OpenSSL_Options.Legacy.LibPath**: レガシーライブラリの場所を設定できます。
 - **oslpNone**: デフォルト。レガシーライブラリはバイナリと同じフォルダーまたは既知のパスにある必要があります。
 - **oslpDefaultFolder**: すべての IDE パーソナリティのライブラリが配置されるべき場所にレガシーライブラリパスを自動設定します。
 - **oslpCustomFolder**: このオプションが選択されている場合は、LibPathCustom プロパティにフルパスを定義してください。
 - **OpenSSL_Options.Legacy.LibPathCustom**: LibPath = oslpCustomFolder の場合、レガシーライブラリのフルパスをここに定義してください。

sgcIndy

- プロパティ **SSLOptions.Legacy** を True に設定します。
- サーバーまたはクライアントを起動する前に、legacy.dll ライブラリが格納されているパスを設定してください。関数 **IdOpenSSLSetOSSLPPath** を使用して、パスを引数として渡します。

OpenSSL | 証明書の検証

OpenSSLを使用し、Verify Certificateオプションを設定すると、次のエラーが表示される場合があります：

SSL との接続エラー。error:80000002:system library::No such file or directory.
イベント OnVerifyPeer を処理し、パラメーター Error の値が 20 の場合、そのエラーは以下を意味します：

X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT_LOCALLY

このエラーの主な原因は、リモートサーバーが提示した1つ以上の証明書が、アプリケーションの証明書ストアに存在しないことです。これを解決するには、RootCertFileプロパティを使用して、CAファイルが配置されているパスを設定できます。ない場合は、例えばmozillaからダウンロードできます。

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

RootCertFile を設定した後、以前のエラーは解消されるはずです。

OpenSSL | 追加関数の読み込み

デフォルトでは Indy は通信の暗号化に必要な最も一般的な OpenSSL 関数を定義していますが、暗号化、署名などに追加の関数が必要な場合があります。IdOpenSSLSetLoadFuncsCallback メソッドを使用して追加の OpenSSL 関数を動的にロードするためのコールバックを割り当てることができます。

IdOpenSSLSetLoadFuncsCallback

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

これは 3 つのパラメータを受け取るコールバックとして機能するプロシージャ型です。

- **hIdSSL**: TIdLibHandle - 読み込まれた SSL ライブラリへのハンドル。
- **hIdCrypto**: TIdLibHandle - ロードされた Crypto ライブラリへのハンドル。
- **FailedLoadList**: TStringList — ロードに失敗した関数のリスト。

このコールバックの目的は、OpenSSL 関数が読み込まれているときにカスタム処理を実行できるようにすることです。例えば、失敗した関数の読み込みをログに記録したり、エラーを処理したりします。

IdOpenSSLSetUnLoadFuncsCallback

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

SSL 関数のアンロードのコールバックとして機能します。OpenSSL ライブラリがアンロードされる際のクリーンアップ処理に役立ちます。

カスタム関数の読み込み方法

コールバックを使用して関数 EVP_PKEY_CTX_set_rsa_padding をロードする簡単な例を以下に示します。

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

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

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

void __fastcall DoOpenSSLUnLoadFuncsCallback() {
    EVP_PKEY_CTX_set_rsa_padding = nullptr;
}

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

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

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


Indy

Indy ライブラリは TCP/UDP/RAW ソケットをサポートするオープンソースのクライアント/サーバー通信ライブラリで、SMTP、POP3、IMAP、NNTP、HTTP、FTP など 100 以上の高レベルプロトコルも含まれています。Indy は Delphi で書かれていますが C++Builder と FreePascal でも利用できます。sgcWebSockets は一部のコンポーネントの基盤として Indy を使用しており、sgcWebSockets の各エディションは Indy ライブラリをそれぞれ異なる形で利用します。

sgcWebSockets は ALPN の使用を必要とする HTTP/2 などのプロトコルをサポートし、openSSL 1.1.1 または openSSL 3.0.0 を使用して TLS 1.3 を使用できます... これらの機能はすべて標準の Indy ライブラリではサポートされていないため、sgcWebSockets Enterprise エディションはこれらの機能をサポートするカスタム Indy ライブラリを含んでいます。IDE から標準の Indy ライブラリをアンインストールする必要があるよう、必要な Indy ファイルはプレフィックス "sgc" を追加してリネームされています。例えば "IdGlobal" ユニットは "sgcIdGlobal" にリネームされます。これにより、両方のバージョンが問題なく共存できます。

以下に、sgcWebSocketsの各エディションで使用されるIndyのバージョンを示します。

sgcWebSockets エディション	Indy バージョン
STANDARD	スタンダード
PROFESSIONAL	スタンダード
ENTERPRISE	カスタム

sgcWebSockets パッケージが分割される前の古いライセンスである「登録済み」ライセンスをお持ちのお客様は、次の sgcWebSockets パッケージバージョンをご利用いただけます：

sgcWebSockets エディション	Indy バージョン
sgcWebSockets	スタンダード
sgcWebSockets 最小	スタンダード
sgcWebSockets min Indy*	カスタム

*sgcWebSockets_min_indy は sgcWebSockets Enterprise エディションと同じです。

カスタム Indy バージョンの使用はソースフォルダーにある「sgcVer.inc」ファイルで定義されています。この Indy バージョンの使用を有効または無効にする「SGC_CUSTOM_INDY」というコンパイラ定義があります。Enterprise Edition をお持ちで、カスタム Indy の使用を無効にしたい場合は、以下のコンパイラ定義を削除してください：

```
{ $DEFINE SGC_CUSTOM_INDY }
```

もちろん、SGC_CUSTOM_INDY を有効にしても、ソースフォルダーに必要なカスタム Indy バージョンのユニットがなければ、このコンパイラ定義は機能しません。

sgcIndy package

カスタム Indy バージョンの使用は、sgcWebSockets コンポーネントに限定されません。一部のユーザーは、SMTP などの標準的な Indy コンポーネントでこのカスタム Indy バージョンの新機能を活用したい場合があります。例えば、メール送信時に TLS 1.3 を使用したり、FTP サーバーを利用したりすることができます。eSeGeCe ALL-ACCESS エディションでは、これらすべての機能を備えた完全な Indy パッケージを追加で提供しています。「sgcIndy パッケージ」と呼ばれるこのパッケージには、openSSL 1.1.1 および openSSL 3.0.0 をサポートした完全な Indy ライブラリが含まれています。まず、IDE にインストールされている現在の Indy ライブラリをアンインストールしてから、このバージョンをインストールしてください。sgcIndy パッケージのインストール手順は、通常の Indy ライブラリと全く同じです（このバージョンではユニット名は変更されません）。

OpenSSL ライブラリを使用する場合は、OpenSSL ライブラリを読み込む前に、グローバル変数 `OPENSSL_API_VERSION` を希望の OpenSSL API バージョンに設定するだけです。このグローバル変数は `IdSSLOpenSSLHeaders` ユニットにあります。

例: openssl 1.1.1 ライブラリを使用する場合

```
OPENSSL_API_VERSION := opSSL_1_1;
```

単一の sgclndy パッケージの使用方法

eSeGeCe ALL-ACCESS と sgclndy パッケージを同じアプリケーションで使用する場合、sgcWebSockets Enterprise バージョンはIndyユニットの名前を変更したカスタムIndyバージョンを使用するため、ソースが重複する可能性があります。つまり、たとえば `sgcldGlobal.pas` と `IdGlobal.pas` のようなユニットが同じアプリケーションでコンパイルされることになります（前者はsgcWebSockets Enterprise パッケージの任意のコンポーネントを使用するときに、後者はftp、smtpなどのsgclndy パッケージの任意のコンポーネントを使用するときに使用されます）。

この動作を回避するために、sgcWebSockets パッケージをインストール済みの sgclndy バージョンを使用するように設定でき、すべてのコンポーネントを引き続き利用できます。これを行うには、以下の手順に従ってください:

1. sgcWebSockets パッケージの Source フォルダにある `sgcVer.inc` ファイルを開きます。
2. コンパイラディレクティブ `SGC_CUSTOM_INDY` を無効にします。このオプションは、`sgcld*.pas` で始まるファイルが存在し、sgcWebSockets Enterprise Packageのコンパイル時に使用する必要があることをコンパイラに伝えます。
3. 以下のコンパイラを有効にします: `SGC_INDY_LIB`。このオプションはコンパイラに sgclndy パッケージがインストールされており、パッケージのコンパイル時に使用される必要があることを伝えます。

```
{ $DEFINE SGC_INDY_LIB }
```

前述の構成を使用すると、sgcWebSockets Enterprise パッケージはインストール済みのsgclndy パッケージを使用し、このパッケージを利用するすべての機能（http/2、IOCP、openssl 3.0 など）が有効になります。

WebSocket イベント

WebSocket 接続には以下のイベントがあります:

OnConnect

新しい接続が確立されたときに発生するイベント。

OnDisconnect

接続が閉じられたときに発生するイベント。

OnError

接続に何らかのエラーがあったときに発生するイベント。

OnMessage

新しいテキストメッセージが受信されたときに発生するイベント。

OnBinary

新しいバイナリメッセージが受信されたときに発生するイベント。

デフォルトでは、sgcWebSocketsはこれらのイベントを発生させるために**非同期**メカニズムを使用します。これらのイベントのいずれかが内部で発生すると、このメッセージをキューに入れ、許可されたときにオペレーティングシステムによってディスパッチされます。この動作は**NotifyEvents**というプロパティを使用して変更できます。デフォルトでは**neAsynchronous**が選択されています。**neNoSync**がチェックされている場合、イベントはメインスレッドと同期せずに発生します（VCLコントロールを更新したり共有リソースにアクセスしたりする必要がある場合は、独自の同期メソッドを実装する必要があります）。

neNoSync の使用が推奨される場合:

1. 非常に短い時間に大量のメッセージを処理する必要があります。
2. プロジェクトがコマンドラインビルド用です（**neNoSync** を設定しないと、イベントが受信されません）。
3. プロジェクトはライブラリです。

そうでない場合は、**neAsynchronous** のデフォルトプロパティ値を使用できます。

WebSocket パラメーター接続

サポート対象

[TsgcWebSocketClient](#)

Java script

新しいWebSocket接続が確立されるときに、クライアントからサーバーに パラメータを渡すことが役立つ場合があります。サーバーにいくつかのパラメータを渡す必要がある場合は、次の プロパティを使用できます。

オプション / パラメータ

デフォルトでは '/' に設定されています。id=1 などのパラメーターを渡す必要がある場合は、このプロパティを '/?id=1' に設定できます。

サーバー側では、次のパラメータを使用して クライアントのパラメータを処理できます:

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

Javascriptを使用すると、 接続URLを使ってパラメータを渡すことができます。例:

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

DLL内部での使用

Dynamic Link Library (DLL) と `sgcWebSockets` を使用する 場合 (またはコンソールアプリケーション)、**NotifyEvents** プロパティを **neNoSync** に設定する必要があります。

Windows サービス

WindowsサービスでsgcWebSocketsライブラリを実行する場合、**NotifyEvents**プロパティを**neNoSync**に設定する必要があります。

```
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;
    }
}
```

Web ブラウザーテスト

TsgcWebSocketServerは、お気に入りのウェブブラウザでWebSocketサーバー接続をテストできる組み込みウェブページを実装しています。

このテストページにアクセスするには、次の URL を入力する必要があります：

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

例: WebSocket サーバーを IP 127.0.0.1 でポート 80 に設定している場合は、以下を入力する必要があります。

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

このページでは、以下の WebSocket メソッドをテストできます：

開く
閉じる
ステータス
送信

WebBrowser HTML テストページを無効にするには、TsgcWebSocketServer.Options.HTMLFiles = false に設定してください。

カスタムサブプロトコル

クライアントは、ハンドシェイクにサブプロトコル名を含めることで、サーバーが特定のサブプロトコルを使用するよう要求できます。指定された場合、接続を確立するには、サーバーは選択されたサブプロトコル値のいずれかをレスポンスに含める必要があります。

独自のサブプロトコルを作成するには、TsgcWSProtocol_Client_Base および TsgcWSProtocol_Server_Base を継承して、カスタムサブプロトコルを作成する必要があります。

認証

サポート対象

TsgcWebSocketServer
TsgcWebSocketHTTPServer
TsgcWebSocketClient

Java script (*URLによる認証のみサポートされています)

WebSocket 仕様には認証メソッドは定義されておらず、ウェブブラウザの実装では新しい WebSocket 接続でカスタムヘッダーを送信することができません。

この機能を有効にするには、次のプロパティにアクセスする必要があります。

Authentication/ Enabled

sgcWebSockets は3種類の WebSocket 認証を実装しています:

セッション: クライアントはユーザー名とパスワードを渡して HTTP GET を実行する必要があります。認証が成功した場合、サーバーはセッション ID で応答します。このセッション ID を使用して、クライアントはそれをパラメータとして渡して WebSocket 接続を開きます。通常の HTTP リクエストを使用して、ユーザーとパスワードをパラメータとして渡してセッション ID を取得することもできます。

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

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

これは WebSocket 接続を使用してサーバーに接続するために使用されるトークンを返します。

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

URL: クライアントはユーザー名とパスワードをパラメータとして渡して WebSocket 接続を開きます。

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

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

Basic: Basic Access Authentication を実装します。VCL WebSocket (サーバーおよびクライアント) と HTTP リクエスト (クライアント Web ブラウザーはこの認証タイプを実装していません) にのみ適用されます。クライアントが接続しようとする時、AUTH BASIC 仕様を使用してヘッダーを送信します。

Authentication/AuthUsers プロパティを使用して、認証済みユーザーのリストを定義できます。各アイテムは次のスキーマに従って定義する必要があります: user=password。例:

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

ユーザーが AuthUsers リストにない場合、クライアントが認証リクエストを送信しない場合などに認証を処理できる **OnAuthentication** というイベントがあります。User と Password パラメータを確認し、正しければ Authenticated 変数を True に設定します。例:

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

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

セキュア接続

サポート対象

TsgcWebSocketServer
TsgcWebSocketHTTPServer
TsgcWebSocketClient
Web ブラウザー

SSL サポートは Indy の実装に基づいているため、この機能を使用するには openssl ライブラリを展開する必要があります。

TsgcWebSocketClient は Microsoft SChannel をサポートしているため、WebSocket クライアントで SChannel オプションを選択した場合、Windows 32 ビットおよび 64 ビット向けに OpenSSL ライブラリをデプロイする必要はありません。

サーバー側

この機能を有効にするには、以下のプロパティを有効にする必要があります。

SSL/ 有効化

定義する必要のある他のプロパティがあります：

SSLOptions/ CertFile/ KeyFile/ RootCertFile: WebSocket 通信を暗号化するには .PEM 形式の証明書が必要です。

SSLOptions/ Password: これはオプションで、証明書にパスワードが設定されている場合にのみ必要です。

SSLOptions/ Port: SSL 接続で使用するポート。

クライアントサイド

この機能を有効にするには、以下のプロパティを有効にする必要があります。

TLS/ 有効

OpenSSL

デフォルトでは、Indy をベースにしたクライアントおよびサーバーコンポーネントは、セキュアな WebSocket サーバーに接続する際に OpenSSL ライブラリを使用します。

Indy は OpenSSL API 1.0.2 のみをサポートしており、API 1.1 はサポートされていません。カスタム Indy ライブラリで sgcWebSockets をコンパイルすると、API 1.1 を使用して TLS 1.3 バージョンを選択できます。OpenSSL_Options プロパティで使用する OpenSSL API を選択してください：

- **osIAPI_1_0:** これはデフォルトの Indy API で、openssl 1.0.2 ライブラリを使用した標準の Indy パッケージを使用できます。
- **osIAPI_1_1:** カスタム Indy ライブラリ（Enterprise Edition）で sgcWebSockets をコンパイルしている場合にのみ選択してください。OpenSSL 1.1.1 ライブラリを使用します。
- **osIAPI_3_0:** カスタム Indy ライブラリ（Enterprise Edition）で sgcWebSockets をコンパイルしている場合にのみ選択してください。openssl 3.0.0 ライブラリを使用します。
 - **ECDHE:** TLS 1.2 の ECDHE を有効にできます（より安全な接続）。

イベント

SSL 設定をカスタマイズするために使用できるイベントが2つあります：

OnSSLGetHandler

このイベントはSSLハンドラーが作成される前に発生します。ここで独自のSSLハンドラーを作成し（TidServerIOHandlerSSLBaseまたはTidIOHandlerSSLBaseから継承する必要があります）、必要なプロパティを設定できます

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

OnSSLAfterCreateHandler

カスタム SSL オブジェクトが作成されていない場合は、OpenSSL ハンドラーを使用してデフォルトで作成されます。必要に応じて SSL ハンドラーのプロパティにアクセスして変更できます。

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

Microsoft SChannel

sgcWebSockets 4.2.6 以降、openssl の代わりに SChannel を使用できます (Windows 7 以降の Windows のみ)。これにより、openssl ライブラリをデプロイする必要がなくなります。TLS 1.0 は Windows 7 からサポートされていますが、Windows 7 で TLS 1.2 などのより新しい実装が必要な場合は、Windows レジストリで TLS 1.1 と TLS 1.2 を有効にする必要があります。

Delphi 2010 Professional Edition（Delphi 7、2007、2009 の場合は Enterprise Edition）が必要です。

HeartBeat

サポート対象

TsgcWebSocketServer
TsgcWebSocketHTTPServer
TsgcWebSocketServer_HTTPAPI
TsgcWebSocketClient

サーバーコンポーネントでは、アクティブなすべての WebSocket 接続に対して x 秒ごとに自動的に ping を送信します。

クライアントコンポーネントでは、x秒ごとにサーバーへ自動的にpingを送信します。

HeartBeat には以下のプロパティがあります:

- **Enabled:** true の場合、ping を送信します
- **Interval:** pingが送信される秒単位の値です。例: 値が10の場合、pingは10秒ごとに送信されます
- **Timeout:** サーバーからの応答を待機する時間です。例: 値が 30 の場合、接続を閉じる前に応答を受信するまで 30 秒待機します。
- **HeartBeatType:** HeartBeatの動作をカスタマイズできます
 - **hbtAlways:** Intervalで定義されたx秒ごとにpingを送信します。
 - **hbtOnlyIfNoMsgRcvInterval:** Intervalプロパティで定義された直近のx秒間にメッセージを受信しなかった場合にのみ、x秒ごとにpingを送信します。

ハートビートのカスタマイズ

クライアントとサーバーのコンポーネントにより、カスタムピングを送信して接続が引き続き生きているかどうかを確認する HeartBeat のカスタマイズが可能です。**OnBeforeHeartBeat** イベントはまさにそのために作られており、カスタムメッセージを送信したり、標準のピングを送信しないようにしたりできます。

使用例: メッセージテキストを30秒ごとにピングとして送信します。

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

WatchDog

サポート対象

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

サーバー

サーバーコンポーネントで、予期しないシャットダウン後に自動的にサーバーを再起動します。60秒ごとにサーバーがアクティブかどうかを確認するには、以下のプロパティを設定します:

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

WatchDog.Monitor では、新しいクライアントがサーバーに接続できるかどうかを確認できます。これは、サーバーへの WebSocket 接続を開こうとする内部クライアントによって行われます。失敗した場合はサーバーを再起動します。クライアントが 10 秒のタイムアウトでサーバーに接続できるかどうかを監視するには、次のプロパティを設定してください:

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

クライアント

クライアントコンポーネントでは、予期しない切断の後に自動的にサーバーへ再接続します。切断後10秒ごとに再接続するには、次のプロパティを設定するだけです:

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

ログ

サポート対象

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

これは WebSocket 接続のデバッグを可能にする便利な機能です。有効にするには次のプロパティにアクセスする必要があります：

LogFile/ Enabled

有効にすると、新しい接続が確立されるたびにテキストファイルに記録されます。サーバーコンポーネントでは、ファイルが存在しない場合は作成されますが、サーバーが閉じるまでアクセスできません。サーバーがアクティブな間にログファイルを開きたい場合は、サーバーを起動する前にログファイルを作成する必要があります。

例:

```
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 メッセージ

WebSocket フレームはマスクされる場合があります。マスクされると ログに記録されたメッセージを読み取ることができません。

プロパティ **LogFile.UnMaskFrames** = True の場合 (デフォルトは true)

- **WebSocket** クライアントが送信したメッセージはマスクなしで保存されます。
- **WebSocket** サーバーが受信したメッセージはマスク済みとマスク解除済みの両方で保存されます (ソケットがバッファを読み取る際にまだメッセージのプロトコルが不明なため、両方を保存します)。

HTTP

サポートされているもの

TsgcWebSocketHTTPServer

TsgcWebSocketHTTPServerは、同じポートを使用してWebSocketとHTTPの接続を処理できるコンポーネントです。HTTPポート（通常はポート80）のみが有効なサーバーをセットアップする必要がある場合に非常に便利です。このコンポーネントは、すべての**TsgcWebSocketServer**機能をサポートし、HTMLページを配信できます。

DocumentRoot プロパティを使用して **HTML ページを静的に提供** できます。例: test.html を "C:\inetpub\wwwroot" ディレクトリに保存し、**DocumentRoot** を "C:\inetpub\wwwroot" に設定すると、クライアントが test.html にアクセスしようとするすると自動的に提供されます:

http://localhost/test.html

または、**HTML やその他のリソースをコードで動的に提供**することもできます。これを行うには、クライアントが新しい HTML ページ、画像、JavaScript ファイルなどをリクエストするたびに発生する **OnCommandGet** イベントがあります。基本的に、クライアントがリクエストしているドキュメントを確認し（**ARequestInfo.Document** を使用）、クライアントにレスポンスを送信する必要があります（**AResponseInfo.ContentText** でレスポンスの内容を送信し、**AResponse.ContentType** でレスポンスの種類を設定し、**AResponseInfo.ResponseNo** に通常 200 のレスポンスコード番号を設定します）。例:

```
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;
    }
}
```


ブロードキャストとチャンネル

サポート対象コンポーネント

[TsgcWebSocketServer](#)

[TsgcWebSocketHTTPServer](#)

[TsgcWebSocketServer_HTTPAPI](#)

Broadcast メソッドはデフォルトで接続中のすべてのクライアントにメッセージを送信しますが、**channels** 引数を使用することで、特定のチャンネルを購読しているクライアントのみにメッセージをブロードキャストするようにフィルタリングできます。

使用例: サーバーにデスクトップとモバイルデバイスという2種類のクライアントが接続している場合、"desktop" と "mobile" という2つのチャンネルを作成することができます。

サーバーの OnConnect イベントでクライアントがモバイルかどうかを識別できる場合は、次のような処理を行うことができます。

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

まず Connection を TsgcWSConnectionServer にキャストしてサブスクリプションメソッドにアクセスし、フィルター条件に合致する場合は desktop チャンネルに登録されます。チャンネルへの登録はどのイベントでも行うことができます。例えば、クライアントにモバイルかどうかを問い合わせ、クライアントの情報を含むメッセージをクライアントからサーバーへ送信してもらうことができます。その後、デスクトップ接続のみにブロードキャストすることができます。

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

100件の接続があり、そのうち30件がモバイルの場合、メッセージは残りの70件にのみ送信されます。

バインディング

サポートされているもの

[TsgcWebSocketServer](#)

[TsgcWebSocketHTTPServer](#)

通常、サーバーには複数の IP があります。WebSocket サーバーを有効にしてリスニングポートを 80 に設定すると、サーバーが起動した際にすべての IP のポート 80 でリッスンしようとしします。そのため IP が 3 つある場合、それぞれのポート 80 にバインドされます。

バインディングを使用すると、サーバーが使用する正確な IP とポートを定義できます。例えば、IP 127.0.0.1 (内部アドレス) と 80.254.21.11 (パブリックアドレス) でポート 80 をリッスンする必要がある場合、サーバーをアクティブにする前に次のように設定できます:

```
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";
}
```

大容量ファイルの送信

サポートコンポーネント

[TsgcWebSocketHTTPServer](#)

[TsgcWebSocketServer_HTTPAPI](#)

HTTP クライアントが **multipart/form-data** ストリームを送信すると、サーバーはそのストリームをメモリ上に保存します。ファイルが大きい場合、サーバーで**メモリ不足**の例外が発生することがあります。この例外を回避するために、サーバーには HTTPUploadFiles プロパティが用意されており、POST ストリームの処理方法（メモリ上またはファイルストリームとして）を設定できます。ストリームをファイルストリームとして処理するように設定すると、受信したストリームはハードディスクに直接保存されるため、メモリの問題を回避できます。サーバーで multipart/form-data ストリームをファイルストリームとして保存するには、次の手順に従ってください。

1. プロパティ **HTTPUploadFiles.StreamType = pstFileStream** を設定します。この設定により、サーバーはストリームをハードディスクに保存します。
2. ファイルストリームとして保存する**最小サイズ（バイト単位）**を設定できます。デフォルト値はゼロで、すべてのストリームがファイルストリームとして保存されます。
3. **SaveDirectory** を使用してストリームの保存先フォルダーを指定します。指定しない場合は、アプリケーションと同じフォルダーに保存されます。
4. クライアントが multipart/form-data を送信すると、コンテンツはバウンダリー内にエンコードされます。プロパティ **RemoveBoundaries** を有効にすると、ストリームの受信が完了した後にバウンダリーの内容が自動的に抽出されます。

サンプルコード

まず新しいサーバーインスタンスを作成し、ストリームがファイルストリームとして保存されるよう設定します。

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

次に、以下の内容で新しい HTML ファイルを作成します。

```
<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>
```

最後に、ウェブブラウザで HTML ファイルを開き、サーバーにファイルを送信します。サーバーは拡張子「.sgc_ps」の新しいファイルストリームを作成し、ストリームの受信が完了するとバウンダリーからファイルを抽出します。

イベント

アップロードファイルのフローをカスタマイズするために使用できる 2 つのイベントがあります（プロパティ HTTPUploadFiles.RemoveBoundaries が有効である必要があります）。

OnHTTPUploadBeforeSaveFile

このイベントはファイルが保存される前に発生し、受信したファイルの名前をカスタマイズできます。

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

OnHTTPUploadAfterSaveFile

このイベントはファイルが保存された後に発生し、保存されたファイルの名前を取得できます。

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

OnHTTPUploadReadInput

このイベントは、デコーダーがファイル入力以外の入力値（例：フォームに名前や日付などの変数がある場合）を受信したときに発生します。

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

圧縮

サポートされているもの

[TsgcWebSocketServer](#)

[TsgcWebSocketHTTPServer](#)

[TsgcWebSocketClient](#)

Chrome などのウェブ ブラウザ

これは大量のデータを送信する必要がある場合に非常に有効な機能です。通常はバイナリメッセージを使用し、Chrome などの一部のブラウザでサポートされている「PerMessage_Deflate」プロトコルで WebSocket メッセージを圧縮します。

この機能を有効にするには、以下のプロパティをアクティブ化する必要があります：

Extensions/ PerMessage_Deflate / Enabled

クライアントがWebSocket サーバーへの接続を試みる際にこのプロパティが有効になっていると、このプロパティを有効にしたヘッダーを送信します。サーバーがこの機能を有効化している場合、このプロトコルを有効にした応答をクライアントに送信し、すべてのメッセージが圧縮されます。サーバーにこの機能がない場合は、すべてのメッセージが圧縮されずに送信されます。

Web ブラウザーでは何も設定する必要はありません。この拡張機能がサポートされている場合は自動的に使用されます。サポートされていない場合、メッセージは圧縮なしで送信されます。

WebSocket メッセージが小さい場合は、このプロパティを有効にしない方がよいでしょう。メッセージの圧縮/解凍に CPU サイクルが消費されるからです。しかし、大量のデータを転送している場合は、メッセージ交換速度の向上が見られます。

Flash

サポート対象

TsgcWebSocketServer
TsgcWebSocketHTTPServer

WebSocketは幅広いWebブラウザでネイティブにサポートされています（<http://caniuse.com/websockets> をご確認ください）。ただし、WebSocketを実装していない古いバージョン（Internet Explorer 6、7、8、9など）も一部存在します。WebSocketを実装していないこれらすべてのブラウザに対して、**Flash Fallback**を有効にできます。

ほぼすべての他のブラウザまたは古いブラウザは、Adobe Flash PlayerをインストールすることでFlashをサポートします。Flash接続をサポートするには、Flashがクロスドメインアクセスをチェックするためにセキュリティ上の理由でこのポートを使用するため、サーバー上で**ポート 843を開く**必要があります。ポート843に到達できない場合は、3秒待ってからサーバーのデフォルトポートへの接続を試みます。

Flash はブラウザが WebSocket をネイティブにサポートしていない場合にのみ適用されます。したがって、サーバー側で Flash フォールバックを有効にしても、ウェブブラウザが WebSocket をネイティブにサポートしている場合は引き続き WebSocket がトランスポートとして使用されます。

Flash フォールバックを有効にするには、サーバーの **FallBack / Flash** プロパティにアクセスして**有効化**する必要があります。さらに2つのプロパティがあります：

1. ドメイン: Flash 接続を単一または複数のドメインに制限する必要がある場合（デフォルトではすべてのドメインが許可されています）。例: これにより swf.example.com ドメインへのアクセスが許可されます。

swf.example.com

2. ポート: flash 接続を単一または複数のポートに制限する必要がある場合（デフォルトではすべてのポートが許可されています）。例: これによりポート 123、456、457、458 へのアクセスが許可されます

123,456-458

Flash 接続はテキストメッセージのみをサポートしており、バイナリメッセージはサポートされていません。

カスタムオブジェクト

サポートされているもの

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

新しい WebSocket 接続が確立されるたびに、sgcWebSockets は識別子、受信/送信バイト数、クライアント IP などのプロパティにアクセスできる [TsgcWSConnection](#) クラスを作成します。また、データベースアクセスやセッションオブジェクトなどのオブジェクトをメモリに保存できる **Data** プロパティもあります。

```
// 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;
    }
}
```

グループ

サポート対象

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

sgcWebSockets は接続されているクライアントの指定されたサブセットにメッセージをブロードキャストする強力な方法を提供します。グループには任意の数のクライアントを含めることができ、クライアントは任意の数のグループのメンバーになれます。グループを明示的に作成する必要はありません。実際には、Groups.Add の呼び出しでその名前を初めて指定した時点でグループが自動的に作成されます。

Groups.Add メソッドを使用してグループにユーザーを追加すると、そのユーザーは現在の接続の期間中、そのグループに向けられたメッセージを受け取ります。

ユーザーの追加と削除

グループにユーザーを追加または削除するには、AddメソッドまたはRemoveメソッドを呼び出し、グループ名とTsgcWSConnectionクラスを渡します。接続が終了したときに、グループからユーザーを手動で削除する必要はありません。

以下の例は Groups.Add メソッドの使用法を示しています。

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

グループへのメッセージ送信

次の例に示すように、グループのすべてのメンバーにメッセージを送信できます。

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

または、「Room」で始まるすべてのグループにメッセージを送信することもできます（Room1、Room2、Room3 が存在する場合、これらのユーザーはメッセージを受け取ります）。

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

イベント

グループとクライアントを処理するために使用できるイベントが2つあります。新しいクライアントがグループに追加されるたびに、またはグループから削除されるたびに発生します。

OnClientAdded
OnClientRemoved

例：メンバーがグループを離れたときにグループにメッセージを送信します。

```
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](#)

*カスタム Indy バージョンが必要です。

Windows用のIOCPは、Indyがデフォルトで行うように接続ごとに1つのスレッドを使用する代わりに、限られたスレッドプールを使用して数千の接続を処理できるAPIです。

Indyサーバーの IOCP を有効にするには、**IOHandlerOptions** プロパティに移動し、IOHandler タイプとして **iohIOCP** を選択します。

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

IOCPThreads は IOCP の非同期リクエスト（オーバーラップ操作）に使用されるスレッドです。デフォルト値はゼロで、プロセッサ数に基づいてスレッド数が計算されます（ただしDelphi 7および2007ではcpucount関数がサポートされていないため、スレッド数は32に設定されます）。

WorkOpThreads は接続が常に同じスレッドで処理されることを望む場合にのみ有効にしてください。IOCP を使用する場合、リクエストはスレッドプールで処理され、同じ接続に対するリクエストも異なるスレッドで処理される可能性があります。すべての接続を同じスレッドで処理したい場合は、WorkOpThreads にこれらのリクエストを処理するために使用されるスレッド数を設定してください。これはサーバーのパフォーマンスに影響を与えるため、この機能が必要な場合にのみ 0 より大きい値を設定することを推奨します。

数千の接続を処理する必要がある場合、Windows サーバーで IOCP を有効にすることをお勧めします。サーバーが最大100の同時接続を処理するだけの場合は、デフォルトの Indy スレッドモデルで問題ありません。

OnDisconnect イベントが発生しない

IOCP はデフォルトの Indy IOHandler とは異なる動作をします。デフォルトの Indy IOHandler では、すべての接続がスレッドで実行され、これらのスレッドは常に実行中で接続がアクティブかどうかを確認します。切断があると短時間で通知されます。

IOCP は異なる動作をします。Indy がデフォルトで行うように 1 スレッド = 1 接続ではなく、すべての接続を処理するスレッドプールがあります。IOCP では、接続がまだ有効かどうかを検出する唯一の方法はソケットへの書き込みを試みることです。エラーが発生した場合、接続が閉じられていることを意味します。切断を検出するには 2 つのオプションがあります:

1. **TsgcWebSocketClient** を使用する場合は、Options プロパティで有効にできます。**CleanDisconnect := True**（デフォルトは無効）。有効にすると、クライアントは切断前に切断をサーバーに通知するメッセージを送信するため、サーバーはこのメッセージを受信し、OnDisconnect イベントが発生します。
2. サーバー側でハートビートを有効にできます。例えば60秒ごとに有効にすると、接続されているすべてのクライアントにpingを送信しようとし、切断されているクライアントがある場合はOnDisconnectが呼び出されます。

EPOLL

Supported by

TsgcWebsocketServer
TsgcWebsocketHTTPServer

*sgcWebSockets Enterprise Editionが必要です。

Linux の EPOLL は、Indy がデフォルトで行うように接続ごとに 1 つのスレッドを使用する代わりに、限られたスレッドプールを使用して何千もの接続を処理できる API です。

Indy サーバーで EPOLL を有効にするには、**IOHandlerOptions** プロパティに移動して IOHandler タイプとして **iohIEPOLL** を選択してください。

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

EPOLLThreads は EPOLL 非同期リクエスト（オーバーラップ操作）に使用されるスレッドです。デフォルト値はゼロで、プロセッサ数に基づいてスレッド数が計算されます（cpucount 関数がサポートされていない Delphi 7 および 2007 では 32 に設定されます）。スレッド数は手動で調整できます。

WorkOpThreads は接続が常に同じスレッドで処理されるようにしたい場合にのみ有効にする必要があります。EPOLL を使用する場合、リクエストはスレッドプールで処理され、同じ接続に対するすべてのリクエストが異なるスレッドで処理される可能性があります。すべての接続を同じスレッドで処理したい場合は、WorkOpThreads にこれらのリクエストを処理するために使用するスレッド数を設定してください。これはサーバーのパフォーマンスに影響し、この機能が必要な場合にのみ 0 より大きい値を設定することが推奨されます。

数千の接続を処理する必要がある場合は、Linux サーバーで EPOLL を有効にすることをお勧めします。サーバーが最大 100 の同時接続しか処理しない場合は、デフォルトの Indy スレッドモデルのままで問題ありません。

OnDisconnect イベントが発生しない

EPOLL はデフォルトの Indy IOHandler とは異なる動作をします。デフォルトの Indy IOHandler では、すべての接続がスレッドで実行され、これらのスレッドは常に実行されて接続がアクティブかどうかを確認します。そのため、切断があった場合、短時間で通知されます。

EPOLL の動作は異なります。Indy がデフォルトで行う 1 スレッド = 1 接続ではなく、すべての接続を処理するスレッドプールがあります。EPOLL では、接続がまだ生きているかどうかを検出する唯一の方法はソケットへの書き込みを試みることです。エラーがある場合は接続が閉じられていることを意味します。切断を検出するには 2 つのオプションがあります:

1. **TsgcWebsocketClient** を使用する場合は、Options プロパティで有効にできます。**CleanDisconnect := True**（デフォルトは無効）。有効にすると、クライアントは切断前に切断をサーバーに通知するメッセージを送信するため、サーバーはこのメッセージを受信し、OnDisconnect イベントが発生します。
2. サーバー側でハートビートを有効にできます。例えば60秒ごとに有効にすると、接続されているすべてのクライアントにpingを送信しようとし、切断されているクライアントがある場合はOnDisconnectが呼び出されます。

Linux接続制限

同時にオープンできる接続数を増やしたい場合は、次のコマンドを使用してください。

```
ulimit -n 10000
```

前のコマンドは開いているファイルディスクリプターの最大数を 10000 に設定します。

ALPN

サポート対象

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

*カスタム Indy バージョンが必要です。

Application-Layer Protocol Negotiation (ALPN) は、アプリケーション層のプロトコルネゴシエーションのための Transport Layer Security (TLS) 拡張です。ALPNを使用すると、アプリケーション層は、追加のラウンドトリップを回避し、アプリケーション層プロトコルから独立した方法で、セキュアな接続上で実行すべきプロトコルをネゴシエートできます。これは、Webページの圧縮を改善し、HTTP/1.xと比較してレイテンシを削減するセキュアなHTTP/2接続に必要です。

クライアント

TLSOptions.ALPNProtocolsで、クライアントがサポートするプロトコルを構成できます。クライアントがサーバーに接続すると、これらのプロトコルは初期TLSハンドシェイクの「Client Hello」で送信され、クライアントがサポートするプロトコルがリストされます。サーバーは、どのプロトコルを使用するかを選択します（存在する場合）。

TsgcWSConnectionClient の ALPNProtocol プロパティにアクセスすることで、サーバーによって選択されたプロトコルを取得できます。

Server

新しい TLS 接続がある場合、OnSSLALPNSelect イベントが呼び出されます。ここでクライアントがサポートするプロトコルのリストにアクセスでき、サーバーはその中からサポートするプロトコルを選択できます。

プロトコルのサポートがない場合、aProtocol は空にできます。

```
// 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 リクエストの転送

サポート対象

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

一部のHTTPリクエストを別のサーバーに転送するようにサーバーを設定できます。これは、複数のサーバーがあり、1つのサーバーだけがパブリックアドレスでリッスンしている場合に非常に便利です。

使用例: サーバーを設定して、/internal へのすべてのリクエストを別のサーバーに転送し、他のリクエストはsgcWebSocketsサーバーで処理させることができます。

イベント**OnBeforeForwardHTTP**を使用して、リクエストされたURLを転送する必要があるかどうかを確認し、転送する場合は転送先のURLを設定します。

例: ドキュメント "/internal" へのすべてのリクエストをサーバー "localhost:8080" に転送したい場合は次のようにします:

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

その他のオプション

HTTPリクエストを転送する場合、以下の追加オプションがあります:

1. デフォルトでは、リクエストは元のドキュメントを使用して転送されます。**使用例:** リクエスト `http://localhost:8080/internal` を内部サーバー `http://localhost:5555` に転送する場合、転送される URL は `http://localhost:5555/internal` になります。ただし、Forward オブジェクトの **Document** プロパティを使用してドキュメントを変更することもできます（デフォルトでは元のリクエストと同じものを使用します）。

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

2. セキュアな HTTP 接続 (HTTPS) を転送する場合は、Forward オブジェクトの **TLSSOptions** プロパティで SSL/TLS オプションをカスタマイズできます。**例:** TLS バージョンを設定します。

```
aForward.TLSSOptions.Version = tls1_2
```

3. 以下のプロパティを使用して HTTP リクエストをカスタマイズできます:

- **QueryParams:** ドキュメントの後のパラメータ。例: 'id=1&user=2'。
- **Host:** リクエストが送信されるサーバーのホストとポート番号を指定します。例: `www.esegece.com:443`
- **Origin:** リクエストの原因となったオリジン (スキーム、ホスト名、ポート)。例: `https://www.esegece.com/document`。
- **LogFilename:** 要求/応答が保存されるファイル名です。
- **NoCache:** リクエストがウェブブラウザのキャッシュを使用しない場合、デフォルトで有効です。
- **CustomHeaders:** リクエストに追加されるカスタムヘッダーのリスト。例: `CustomHeaders.Add('X-ReverseProxy-Host: http://127.0.0.1:8888/test');`

Quality Of Service

サポートされているもの

TsgcWSPServer_sgc
TsgcWSPClient_sgc
TsgcWSPClient_MQTT
Java script

SGC デフォルトプロトコル と MQTT はメッセージ配信のQoS（サービス品質）を実装しています。3種類があります。

レベル 0:「最大1回」。メッセージは基盤となるTCP/IPネットワークのベストエフォートに従って配信されます。メッセージの損失や重複が発生する可能性があります。このレベルは、次の読み取りがすぐに公開されるため、個々の読み取りが失われても問題ない周囲センサーデータなどに使用できます。

レベル 1:「少なくとも 1 回」。メッセージは確実に届きますが、重複する場合があります。

レベル 2:「正確に 1 回」。メッセージは正確に 1 回届くことが保証されます。このレベルは、例えば重複または欠落したメッセージが誤った請求につながる可能性がある課金システムで使用できます。

レベル 0

メッセージは基盤となる TCP/IP ネットワークのベストエフォートに従って配信されます。レスポンスは期待されず、プロトコルで再試行セマンティクスは定義されていません。メッセージはサーバーに一度だけ届くか、全く届かないかのどちらかです。

以下の表は QoS レベル 0 のプロトコルフローを示しています。

クライアント	メッセージと方向	サーバー
QoS = 0	PUBLISH ----->	Action: サブスクライバーにメッセージを公開する

レベル 1

サーバーによるメッセージの受信は ACKNOWLEDGEMENT メッセージによって確認されます。通信リンクまたは送信デバイスの障害が識別された場合、または指定された時間内に確認メッセージが受信されなかった場合、送信側はメッセージを再送します。メッセージはサーバーに少なくとも1回届きます。

QoSレベル1のメッセージには、メッセージにMessage IDがあります。

下の表は QoS レベル 1 のプロトコルフローを示しています。

クライアント	メッセージと方向	サーバー
QoS = 1 Message ID = x Action: メッセージを 保存する	PUBLISH ----->	Actions: • メッセージを 保存 • サブスクライバーに メッセージをパブリッシュ • メッセージを 削除
アクション: メッセージを破棄する	ACKNOWLEDGEMENT <-----	

クライアントが ACKNOWLEDGMENT メッセージを受信しない場合（アプリケーションで定義された時間内に、または障害が検出されて通信セッションが再起動された場合）、クライアントは PUBLISH メッセージを再送信できます。

レベル 2

QoSレベル1を超える追加のプロトコルフローにより、重複メッセージが受信側アプリケーションに配信されないことが保証されます。これは最高レベルの配信であり、重複メッセージが受け入れられない場合に使用します。ネットワークトラフィックは増加しますが、メッセージ内容の重要性のため、通常は許容されます。

QoS レベル 2 のメッセージには、メッセージ内にメッセージ ID があります。

以下の表は、QoSレベル2のプロトコルフローを示しています。受信者がPUBLISHフローをどのように処理すべきかについて、2つのセマンティクスが利用可能です。

クライアント	メッセージと方向	サーバー
QoS = 2 メッセージ ID = x Action: メッセージを 保存する	PUBLISH ----->	アクション: メッセージを保存します
	PUBREC <-----	メッセージ ID = x
メッセージ ID = x	PUBREL ----->	アクション: - サブスクライバーへのメッセージのパブリッシュ - メッセージの削除
アクション: メッセージを破棄する	確認応答 (ACKNOWLEDGEMENT) <-----	メッセージ ID = x

障害が検出された場合、または定義された期間後に、最後の未確認プロトコルメッセージからプロトコルフローが再試行されます。追加のプロトコルフローにより、メッセージがサブスクライバーに正確に1回配信されることが保証されます。

キュー

サポートされているもの

[TsgcWSPServer_sgc](#)

[TsgcWSPClient_sgc](#)

Java script

[SGC Default Protocol](#) はキューを実装して発行されたメッセージに永続性を追加します (**発行メッセージ**のみに対応)

レベル 0: メッセージはサーバーでキューに入れられません

レベル 1: サーバーには最後のメッセージのみがキューに入れられ、クライアントが新しいチャンネルをサブスクライブするかサーバーに接続するたびに送信されます。

レベル 2: すべてのメッセージがサーバーでキューに入れられ、クライアントが新しいチャンネルにサブスクライブするかサーバーに接続するたびに送信されます。

レベル 0

メッセージはサーバーによってキューイングされません

以下の表はキューレベル 0 のプロトコルフローを示しています。

クライアント	メッセージと方向	サーバー
Queue = 0	PUBLISH ----->	Action: サブスクライバーにメッセージを公開する

レベル 1

キュー レベル 1 のメッセージはサーバーに保存され、このチャンネルで他のメッセージが保存されている場合、それらは削除されます。

以下の表は Queue レベル 1 のプロトコルフローを示しています。

クライアント	メッセージと方向	サーバー
Queue = 1	PUBLISH ----->	アクション: - このチャンネルの すべてのメッセージを削除する - チャンネルごとに 最後のメッセージを保存する
アクション: メッセージを処理する	NOTIFY <-----	Action: 新しいクライアントがこのチャンネルにサブスクライブするたびに、最後のメッセージが送信されます。

これは、パブリッシャーが「例外による報告」ベースでメッセージを送信し、メッセージ間にしばらく時間が空く可能性がある場合に便利です。これにより、新しいサブスクライバーは保持された、つまり最後に既知の正常な値を持つデータを 即座に受信できます。

レベル 2

キューレベル 2 のすべてのメッセージ はサーバーに保存されます。

以下の表は、Queueレベル2プロトコル のフローを示しています。

クライアント	メッセージと方向	サーバー
Queue = 2	PUBLISH ----->	アクション: メッセージを保存します
アクション: メッセージを処理する	NOTIFY <-----	Action: 新しいクライアントがこのチャンネルをサブスクライブするたびに、すべてのメッセージが送信されます。

Transactions

サポートされているもの

[TsgcWSPServer_sgc](#)

[TsgcWSPClient_sgc](#)

Java script

sgcWebSockets SGC プロトコルはトランザクションメッセージングをサポートしています。クライアントがトランザクションをコミットすると、クライアントが送信したすべてのメッセージがサーバー側で処理されます。クライアントが呼び出すメソッドは3つあります:

StartTransaction

サーバー側で新しいトランザクションを作成し、このメソッドの後にクライアントからサーバーに送信されるすべてのメッセージは、クライアントが Commit または Rollback を呼び出すまでサーバー側でキューに入れられます。

クライアント	メッセージと方向	サーバー
チャンネル = X	STARTTRANSACTION ----->	Action: 指定チャンネルのすべてのメッセージを格納する新しいキューを作成します。
チャンネル = X	PUBLISH ----->	Action: メッセージはサーバー側に保存されます。
Action: クライアントがメッセージ送信の確認を受け取ります	確認応答 < -----	アクション: メッセージが保存されたため、サーバーはクライアントに確認応答を返します。
....		

Commit

クライアントが Commit を呼び出すと、サーバーでキューに入れられたすべてのメッセージが処理されます。

クライアント	メッセージと方向	サーバー
チャンネル = X	COMMIT ----->	Action: Transactionによってキューに入れられたすべてのメッセージを処理します

RollBack

クライアントが RollBack を呼び出すと、サーバーによってキューに入れられたすべてのメッセージが削除され、サーバー側で処理されません。

クライアント	メッセージと方向	サーバー
チャンネル = X	ROLLBACK ----->	アクション: トランザクションによってキューに入れられたすべてのメッセージを削除

TCP 接続

サポート対象

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

デフォルトでは、sgcWebSocket はプロトコルとして WebSocket を使用しますが、クライアントおよびサーバーコンポーネントでプレーン TCP プロトコルを使用することもできます。

クライアント コンポーネント

WebSocket プロトコルを無効にします。

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

サーバーコンポーネント

OnUnknownProtocol イベントを処理し、Transport を trpTCP に設定して接続を受け入れます。

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

クライアントがサーバーに接続すると、この接続は TCP として定義され、WebSocket の代わりにプレーン TCP プロトコルを使用します。プレーン TCP 接続はテキスト メッセージとバイナリ メッセージを区別しないため、受信したすべてのメッセージは OnBinary イベントで処理されます。

メッセージの終わり

メッセージが大きい場合、フラグメント化されて受信されることがあります。どのバイトを探すかを指定することでメッセージの終わりを検出するメソッドがあります。例: STOMP プロトコルでは、すべてのメッセージはバイト 0 と 10 で終わります。

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

SubProtocol

サポート対象

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

WebSocket はシンプルなサブプロトコルネゴシエーションを提供します。基本的にリクエストがサポートするプロトコル名を含むヘッダーを追加します。これらのプロトコルが受信され、受信者がそのうちの1つをサポートしている場合、サポートされているサブプロトコルを含むレスポンスを送信します。

sgcWebSockets はいくつかのサブプロトコルをサポートしています: [MQTT](#)、[WAMP](#) など。RegisterProtocol を呼び出してサブプロトコル名を引数として渡すだけで、非常に簡単な方法で独自のサブプロトコルを実装できます。

例: サブプロトコル「Test 1.0」を実装するサーバーに接続する必要があります。

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

1 つの接続で複数のプロトコルを使用するには、ブローカープロトコル（サーバーおよびクライアント）コンポーネントを使用できます。クライアント/サーバーとプロトコルの間にブローカーを配置するだけです。例: 単一接続で SGC プロトコルとファイルプロトコルを使用する。

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

ブローカープロトコルがサーバー/クライアントとプロトコルの間に接続されている場合、**OnConnect** および **OnDisconnect** イベントはブローカーコンポーネントで発生します (サーバーまたはクライアントコンポーネントではなく)。

スロットル

サポートされているもの

[TsgcWebSocketServer](#)

[TsgcWebSocketHTTPServer](#)

[TsgcWebSocketClient](#)

帯域幅スロットリングはサーバーおよびクライアントコンポーネントでサポートされており、有効にすると、ソケットによって毎秒送受信されるビット数を制限できます。Indyはブロッキング方式を使用するため、クライアントが読み取りを制限している場合、未読データはクライアントソケット内にとどまり、サーバーは新しいデータをクライアントへ書き込めなくなります。クライアントがデータを 読み取るのが遅いほど、サーバーが新しいデータを書き込めるのも遅くなります。

Server-sent Events (プッシュ通知)

サポート対象

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

SSE は WebSocket の一部ではなく、サーバーからプッシュ通知を受け取るための HTTP 接続を開くための API を定義します。

SSEは従来のHTTP上で送信されます。つまり、動作させるために特別なプロトコルやサーバー実装を必要としません。さらに、Server-Sent Eventsには、自動再接続、イベントID、任意のイベントを送信する機能など、WebSocketには設計上欠けているさまざまな機能があります。

イベント

- **Open:** 新しい SSE 接続が開かれたとき。
- **Message:** クライアントが新しいメッセージを受信したとき。
- **エラー:** 切断などの接続エラーが発生した場合。

JavaScript API

イベントストリームをサブスクライブするには、EventSource オブジェクトを作成し、ストリームの URL を渡します:

```
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);
```

サーバーからアップデートがプッシュされると、onmessage ハンドラーが発生し、新しいデータが e.data プロパティで利用可能になります。接続が閉じられた場合、ブラウザは約 3 秒後に自動的にソースに再接続します（これはデフォルトの再試行間隔で、サーバー側で変更できます）。

フィールド

次のフィールド名が仕様によって定義されています:

イベント

イベントのタイプ。これが指定されている場合、指定されたイベント名のリスナーに対してブラウザでイベントがディスパッチされます。ウェブサイトは addEventListener() を使用して名前付きイベントをリスンします。メッセージにイベント名が指定されていない場合は onmessage ハンドラーが呼び出されます。

data

メッセージのデータフィールド。EventSourceが data:で始まる複数の連続した行を受け取った場合、それらを連結し、各行の間に改行文字を挿入します。末尾の改行は削除されます。

id

EventSource オブジェクトの最後のイベント ID 値を設定するためのイベント ID。

リトライ

イベントを送信しようとするときに使用する再接続時間。整数値で、ミリ秒単位の再接続時間を指定する必要があります。整数以外の値が指定された場合、このフィールドは無視されます。

その他のフィールド名はすべて無視されます。

複数行の文字列には #10 を改行として使用してください。

使用例：

クライアントにメッセージを送信する必要がある場合は、WriteData メソッドを使用してください。

```
// 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

サポートされているもの

[TsgcWebSocketServer](#)

[TsgcWebSocketHTTPServer](#)

[TsgcWebSocketLoadBalancerServer](#)

ロードバランシングにより、複数のバックエンドサーバー間で作業を分散できます。新しいクライアントが接続リクエストを行うたびに、ロードバランサーサーバー（バックエンドサーバーに接続されている）に接続し、クライアントがサーバーに接続するために使用するホスト、ポートなどの情報を含む接続文字列を返します。例えば4台のサーバーがある場合、この方法により全サーバーのほぼ同数の接続とほぼ同等のワークロードが実現できます。

クライアントがすべてのサーバーのすべてのクライアントにメッセージを送信したい場合は、Broadcast メソッドを使用するだけで、メッセージはロードバランサーサーバーに接続されているすべてのサーバーにブロードキャストされます。

この機能を有効にするには:

1. [TsgcWebSocketLoadBalancerServer](#) コンポーネントをドロップし、リスニングポートを設定してActiveをTrueにします。
2. サーバーとクライアントコンポーネントには LoadBalancer というプロパティがあり、ロードバランサーサーバーのホストとポートを設定し、Enabled を True に設定する必要があります。

このコンポーネントは **WebSocket** と **HTTP** プロトコルのロードバランシングを可能にします。

ファイル

サポートされているもの

[TsgcWSPServer_sgc](#)

[TsgcWSPClient_sgc](#)

このプロトコルは、クライアントからサーバーへ、またはサーバーからクライアントへのファイル送信を簡単な方法で可能にします。非常に小さなファイルから大きなファイルまで、低メモリ使用量で送信できます。設定可能な項目：

1. パケットサイズ（バイト単位）。
2. カスタム チャンネルを使用して、サブスクライブしているクライアントのみにファイルを送信します。
3. ファイルの送受信の進行状況。
4. 受信したファイルの認証。
5. 送信パケットの確認応答。

プロキシ

サポートされているもの

[TsgcWebSocketClient](#)

クライアント WebSocket コンポーネントは HTTP プロキシを通じた WebSocket 接続をサポートします。プロキシ接続を有効にするには、以下のプロパティを有効にする必要があります。

プロキシ / 有効

True に設定すると、以下を設定できます。

Host: プロキシサーバーアドレス。

Port: プロキシサーバーのポート

UserName/Password: プロキシへの接続認証。必要な場合のみ。

ProxyType: 以下のプロキシがサポートされています：

- HTTP
- Socks4
- Socks4A
- Socks5

SOCKS プロキシを設定するには SOCKS プロパティにアクセスして Enabled を True に設定します。

フラグメント化されたメッセージ

サポート対象

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

デフォルトでは、sgcWebSocketsライブラリを使用してストリームが送信される場合、すべてのデータを単一のパケットで送信するか、すべてのパケットをバッファリングし、最後のパケットが受信されると、OnBinaryメッセージイベントが呼び出されます。

この動作は **Options.FragmentedMessages** プロパティによってカスタマイズでき、次の値を受け入れます:

1. frgOnlyBuffer: これがデフォルト値です。パケットメッセージがバッファリングされ、ストリーム全体が受信されたときにのみ OnBinary メッセージイベントが呼び出されることを意味します。
2. frgOnlyFragmented: これはフラグメント化されたパケットごとに OnFragmented イベントのみが呼び出されることを意味します。
3. frgAll: これは OnFragmented イベントが受信されたすべてのパケットに対して呼び出され、完全なストリームが受信されたときにも呼び出されることを意味します。

OnFragmented イベントは、大きなストリームを送信して受信者が転送の進捗を表示する必要がある場合に役立ちます。

使用例: クライアントはサーバーに 1,000,000 バイトのストリームを送信する必要があり、サーバーは受信した 1,000 バイトごとに進捗を表示したいとします。

クライアントは、パケットサイズ 1000 で writedata メソッドを使用してストリームを送信します。

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

サーバーは Options.FragmentedMessages := frgAll を設定し、ストリームの進捗を受信するために OnFragmented イベントを処理します

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

TsgcWebSocketClient

クライアントWebSocketコンポーネントです。任意のRFC 6455 WebSocketサーバーに接続し、テキストおよびバイナリメッセージを送受信します。

はじめに

TsgcWebSocketClientはクライアントWebSocketコンポーネントを実装しており、WebSocketサーバーに接続できます。このコンポーネントを設定するには、以下の手順に従ってください:

1. フォームに **TsgcWebSocketClient** コンポーネントをドロップします。
2. 利用可能なWebSocketサーバーに接続するために**Host**と**Port**（デフォルトは80）を設定します。**URL**プロパティを設定すると、Host、Port、ParametersなどがURLから更新されます。例: wss://127.0.0.1:8080/ws/ は次のようになります:

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

3. TLS が必要かどうかを選択できます（セキュア接続）。デフォルトでは有効になっていません。
4. HTTP プロキシサーバーを通じて接続できます。プロキシのプロパティを定義する必要があります。
Host: プロキシサーバーのホスト名。
Port: プロキシサーバーのポート番号。
Username: 認証用ユーザー名。匿名の場合は空白のままにします。
パスワード: 認証用のパスワード。匿名の場合は空白のままにしてください。
5. サーバーが圧縮をサポートしている場合、圧縮を有効にして送信するメッセージを圧縮できます。
6. 許可される**Specifications**を設定します。デフォルトでは、すべての仕様が有効です。
RFC6455: 標準かつ推奨の WebSocket 仕様です。
Hixie76: 常にfalse
7. Active = true に設定して新しい WebSocket 接続を開始します。

リファレンス

- プロパティ — コンポーネントで使用可能な 23 のプロパティ。
- メソッド — 公開メソッド 22 件。
- イベント — 処理できるイベント 16 件。

ガイド

最も一般的な使用方法

- 接続
 - [WebSocket サーバーへの接続方法](#)
 - [クライアント接続を開く](#)
 - [クライアント接続を閉じる](#)
 - [接続をアクティブに維持する](#)
 - [突然の切断](#)
 - [TCP サーバーへの接続](#)
 - [WebSocket リダイレクション](#)
- セキュアサーバー
 - [セキュアサーバーへの接続](#)
 - [OpenSSL 証明書](#)

- [SChannel 証明書](#)
- [SChannel 接続情報の取得](#)
- [メッセージを送信する](#)
 - [テキストメッセージを送信](#)
 - [バイナリメッセージを送信](#)
- [メッセージを受信する](#)
 - [テキストメッセージを受信](#)
 - [バイナリメッセージを受信](#)
- [認証](#)
 - [クライアント認証](#)
- [Other](#)
 - [Client Exceptions](#)
 - [Client WebSocket HandShake](#)
 - [Client Register Protocol](#)
 - [Client Proxies](#)

TsgcWebSocketClient | WebSocket サーバーへの接続

URL プロパティ

WebSocket サーバーへの接続で最も簡単な方法は、**URL** プロパティを使用して **Active = true** を設定することです。

使用例: セキュア接続を使用して www.esegece.com に接続します。

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

ホスト、ポート、パラメーター

Host プロパティと port プロパティを使用して WebSocket サーバーに接続できます。

例: セキュアな接続を使用して www.esegece.com に接続する

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


TsgcWebSocketClient | クライアントの接続を開く

クライアントがサーバーへの接続を構成した後、新しい接続を開くには3つの異なる オプションがあります。

Active プロパティ

新しい接続を開く最も簡単な方法は、Active プロパティを true に設定することです。これにより、コンポーネントの設定を使用してサーバーへの接続が試みられます。

Active プロパティを false に設定すると、アクティブな場合は接続を閉じます。

このメソッドは呼び出し元と同じスレッドで実行されます。メインスレッドから呼び出すと、メソッドはアプリケーションのメインスレッドで実行されます。

接続を開く

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

Active = true を呼び出した場合、クライアントがまだ接続中の可能性があるため、**まだサーバーヘータを送信できません**。まず OnConnect イベントが発生するまで待つ必要があります、その後でサーバーへ メッセージの送信を開始できます。

接続を閉じる

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

Active = false を呼び出した場合、**このコードの直後に接続がすでに閉じられているとは限りません**。そのため、OnDisconnect イベントが発生するまで 待つ必要があります。

Start/Stop メソッド

サーバーに接続/切断するために Start() または Stop() を呼び出すと、呼び出しはセカンダリスレッドで実行されるため、呼び出されたスレッドをブロックしません。サーバーに接続して以降のコードを続けて実行したい場合はこのメソッドを使用してください。

接続を開く

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

Start() を呼び出しても、クライアントがまだ接続中の可能性があるため、**サーバーにデータを送信することはまだできません**。まず OnConnect イベントが発生するまで待機してから、サーバーへのメッセージ送信を開始してください。

接続を閉じる

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

Stop() を呼び出した後、**このコードの直後に接続がすでに閉じられているとは限りません**。OnDisconnect イベントが発火するまで待機する必要があります。

接続/切断メソッド

Connect() または Disconnect() を呼び出してサーバーへの接続を開く/閉じる場合、呼び出しは呼び出されたスレッドと同じスレッドで実行されますが、プロセスが完了するまで待機します。プロセスが完了するまでの最大待機時間を定義するために Timeout を設定する必要があります（デフォルトは 10 秒）。

使用例: サーバーに接続して最大 5 秒間待機する

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

Connect() メソッドが成功した結果を返した場合、接続が生きているため、すでにサーバーにメッセージを送信できます。

例: サーバーから切断して最大 10 秒間待機します

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

Disconnect()メソッドが成功した結果を返した場合、それは接続が既に閉じられていることを意味します。

OnBeforeConnect イベントを使用して、クライアントがサーバーへの接続を試みる前にサーバー接続プロパティをカスタマイズできます。

TsgcWebSocketClient | クライアント接続の切断

接続は `Active` プロパティ、`Stop` または `Disconnect` メソッドを使用して閉じることができます。詳細は [クライアント接続の開始](#) を参照してください。

CleanDisconnect

接続が閉じられる場合、接続が閉じられていることを示すクローズメッセージを送信することで相手ピアに通知できます。この機能を有効にするには、`Options.CleanDisconnect` プロパティを `true` に設定してください。このプロパティが有効の場合、接続が閉じられる前に、クライアントが接続を閉じることをサーバーに通知するために `Close` メッセージが送信されます。

切断

[TsgcWSConnection](#) には `Disconnect()` というメソッドがあり、ソケットレベルで接続を切断できます。このメソッドを呼び出すと、サーバーからの応答を待たずにソケットが直接切断されます。このメソッドでクローズコードを送信できます。

閉じる

[TsgcWSConnection](#) には `Close()` というメソッドがあり、接続を閉じるようサーバーにメッセージを送信できます。サーバーがこのメッセージを受信すると、接続を閉じる必要があり、クライアントは接続が閉じられたという通知を受け取ります。このメソッドでクローズコードを送信できます。

TsgcWebSocketClient | Client Keep Connection Open

クライアントがサーバーに接続すると、信号の弱さ、接続エラーなどにより接続が切断される場合があります。接続をアクティブに保つための2つのプロパティがあります。

HeartBeat

HeartBeat プロパティを使用すると、X 秒ごとに **Ping** を送信して接続を維持できます。サーバーによっては、ピア間でデータが交換されない場合に TCP 接続を閉じるものがあります。HeartBeat はこの問題を特定の間隔で ping を送信することで解決します。通常これで接続をアクティブに維持するのに十分ですが、サーバーから X 秒後にレスポンスが受信されない場合に接続を閉じるには Timeout 間隔を設定できます。

使用例： 30秒ごとにピングを送信します。

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

OnBeforeHeartBeat というイベントがあり、HeartBeat の動作をカスタマイズできます。デフォルトでは、HeartBeat が有効な場合、クライアントは HeartBeat.Interval プロパティで設定した X 秒ごとに WebSocket ping を送信します。

OnBeforeHeartBeat には Handled というパラメータがあり、デフォルトでは false です。これは、フローが TsgcWebSocketClient コンポーネントによって制御されることを意味します。値を True に設定すると、ping は送信されず、Connection クラスを使用してカスタムメッセージを送信できます。

WatchDog

WatchDog が有効な場合、クライアントが切断を検出すると、接続が再確立されるまで WatchDog は X 秒ごとに再接続を試みます。

使用例: 切断後に無制限の試行回数で 10 秒ごとに再接続します。

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

OnBeforeWatchDog イベントを使用して、クライアントが接続を試みるサーバーを変更できます。**例:** 3 回再試行した後、クライアントがサーバーに接続できない場合は、セカンダリサーバーへの接続を試みます。

Handled プロパティを True に設定すると、クライアントは再接続を試みません。

TsgcWebSocketClient | 切断の検出

接続が確立された後、どのピアもデータを送信しない場合、ネットワーク上にパケットは送信されません。TCP はアイドルプロトコルであるため、接続がまだアクティブであると見なされます。

切断理由

- ・ **アプリケーションの終了:** プロセスが終了すると、通常は FIN パケットを送信して接続が閉じられたことをもう一方のピアに通知します。しかし、プロセスがクラッシュした場合、このパケットがもう一方のピアに送信される保証はありません。
- ・ **デバイスが閉じる:** デバイスが閉じる場合、通常はこれについての通知はありません。
- ・ **ネットワークケーブルが抜かれた場合:** ネットワークケーブルが抜かれた場合、それはルーターが閉じるのと同じです。データが転送されていないため、接続は閉じられません。
- ・ **ルーターからの信号が失われた場合:** アプリケーションがルーターからの信号を失っても、接続は維持されます。

ハーフオープン切断の検出

以下のメソッドを使用して切断の検出を試みることができます。

2番目の接続

2 番目の接続を開いて接続を試みることもできますが、これにはいくつかの欠点があります。より多くのリソースを消費し、新しいスレッドを作成するなどの問題があります。また、相手ピアが再起動した場合、2 番目の接続は機能しますが、最初の接続は機能しません。

他のピアに Ping を送信します。

ハーフオープン接続で ping やメッセージを送信しようとしても、エラーが発生しないことがわかります。

TCP ソケットレベルで KeepAlive を有効にする

TCP キープアライブパケットは、接続の現在のシーケンス番号より 1 少ないシーケンス番号が設定された単純な ACK です。これらの ACK のいずれかを受信したホストは、現在のシーケンス番号の ACK で応答します。キープアライブは、接続のリモート端にあるコンピューターがまだ利用可能かどうかを確認するために使用できます。TCP キープアライブは、TCP 接続でデータまたは上位レベルのキープアライブが送信されていない場合、TCPKeepAlive.Time ごとに 1 回（デフォルトは 7,200,000 ミリ秒または 2 時間）送信できます。キープアライブへの応答がない場合、TCPKeepAlive.Interval 秒ごとに 1 回繰り返されます。KeepAliveInterval のデフォルトは 1000 ミリ秒です。

接続ごとの KeepAlive を有効にして、TCP プロトコルが接続がアクティブかどうかを確認できるようにすることができます。これは切断されたドロップを検出したい場合（たとえば、ネットワークケーブルを抜いたとき）に推奨される方法です。

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

TsgcWebSocketClient | TCPサーバーに接続する

TsgcWebSocketClient は WebSocket サーバーに接続できますが、プレーン TCP サーバーにも接続できます。

URL プロパティ

TCP サーバーに接続する最も簡単な方法は、**URL** プロパティを使用して **Active = true** に設定することです。

例: 127.0.0.1 ポート 5555 に接続

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

ホスト、ポート、パラメーター

HostプロパティとPortプロパティを使用してTCPサーバーに接続できます。

例: 127.0.0.1 ポート 5555 に接続

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

TsgcWebSocketClient | 接続 TIME_WAIT

クライアントがサーバーから切断を開始すると、切断の状態を伝えるためにクライアントとサーバーの間でやり取りが行われます。プロセスが完了すると、クライアントのソケット接続は一定期間 TIME_WAIT 状態に入ります。これは正常な動作であり、Windows オペレーティング システムではデフォルトで約 4 分です。

この動作を軽減または排除することができますが、以下の代替手段を使用する際は注意が必要です。

REGEDIT

Windows RegeditでTIME_WAIT値を削減できます。

1. レジストリエディタを開き、HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TCPIP\Parameters レジストリサブキーにアクセスします。
2. **TcpTimedWaitDelay**という名前の新しいREG_DWORD値を作成します
3. 秒単位で値を設定します。例: 値を 5 に設定すると、TIME_WAIT は最大 5 秒間待機します。
4. 保存してシステムを再起動します。

LINGER

TIME_WAIT 状態を回避するもう一つの方法は、ソケットオプション SO_LINGER を使用することです。有効にすると、接続を正常にクローズする代わりにクライアントが接続をリセットするため、TIME_WAIT 状態が回避されます。

LingerState プロパティを使用してこのオプションを有効にできます。デフォルト値は -1 です。値を 0 に設定すると、タイムアウトなしでソケットから切断する際に接続がリセットされます。このオプションはおそらく最も推奨されず、最終手段としてのみ使用するべきです。

TsgcWebSocketClient | WebSocket リダイレクト

クライアントが WebSocket サーバーに接続すると、サーバーは HTTP レスポンスコード 30x を返すことがあります。レスポンスコードが 301 の場合、場所が永続的に移動されたことを意味し、新しい URL が Location HTTP ヘッダーに提供されます。

WebSocketクライアントはリダイレクトを自動的に処理します。そのため、サーバー応答にリダイレクトが含まれていることを検出すると、現在の接続を切断し、新しいLocation URLへの接続を試みます。

使用例

1. クライアントは最初に ws://127.0.0.1:5000 への接続を試みます
2. サーバーは 301 のレスポンスコードを返し、値 ws://80.50.1.2:3000 の Header Location を含みます。
3. クライアントはサーバーからのレスポンスを読み取り、リダイレクトであることを検出してLocationを読み取ります
 1. 現在の接続を最初に切断します。
 2. Location ヘッダーの値 (ws://80.50.1.2:3000) で URL プロパティを更新します。
 3. 新しいサーバーに接続します。

TsgcWebSocketClient | セキュアサーバーへの接続

TsgcWebSocketClient は、セキュアおよび非セキュアな接続を使用して WebSocket サーバーに接続できます。

URLプロパティ、またはHost / Portプロパティを使用してセキュア接続を構成できます。[WebSocketサーバーへの接続](#)を参照してください。

TLSTOptions

TLSTOptions プロパティには**セキュア接続をカスタマイズするためのプロパティ**があります。最も重要なプロパティは **version** で、**TLS プロトコルのバージョン**を指定します。通常、**TLS プロパティ**を **true** に設定し、**TLSTOptions.Version** を **tlsUndefined** に設定するだけで、大多数の WebSocket サーバーに対して十分です。

TLSTOptions.Version では、サーバーへの接続に使用する TLS バージョンを設定するか、クライアントが利用可能なすべてのバージョンから TLS バージョンを交渉できるようにできます（値が **tlsUndefined** の場合のデフォルト）。

TLSプロトコルに関するサーバーへの接続を試みる際のエラーが発生する場合、おそらくサーバーが設定されているものより**新しいTLSバージョンを必要**としている可能性が高いです。

TLSTOptions.IOHandler が **iohOpenSSL** に設定されている場合、すべての TLS を処理するライブラリである **OpenSSL** ライブラリをデプロイする必要があります。[OpenSSL](#) に関する以下の記事を参照してください。

TLSTOptions.IOHandler が **iohSChannel** に設定されている場合、ライブラリをデプロイする必要はありません（Windows のみサポート）。

TsgcWebSocketClient | Certificates

OpenSSL

サーバーがクライアントに SSL 証明書を使用した接続を要求する場合は、TsgcWebSocketClient の TLSOptions プロパティを使用して証明書ファイルを設定します。証明書は PEM 形式である必要があります。そのため、異なる形式の場合は、まず PEM に変換する必要があります。

OpenSSL ライブラリを通じた接続には TLSOptions.IOHandler = iohOpenSSL が必要です。

次のプロパティを構成します:

- **CertFile:** PEM 形式の証明書へのパス。
- **KeyFile:** 証明書の秘密鍵へのパスです。
- **RootCertFile:** 証明書のルートへのパスです。
- **Password:** 証明書がパスワードで保護されている場合は、ここにシークレットを設定します。

TsgcWebSocketClient | 証明書 SChannel

サーバーがクライアントにSSL証明書を使用して接続することを要求する場合、TsgcWebSocketClientのTLSOptionsプロパティを使用して証明書ファイルを設定してください。

SChannel を通じた接続には TLSOptions.IOHandler = iohSChannel が必要です。

SChannel は 2 種類の証明書認証をサポートしています：

1. **PFX証明書**を使用する
2. Windows システムにすでにインストールされている証明書の**ハッシュ証明書**を設定します。

PFX 証明書

PFX 証明書は証明書と秘密鍵を含むファイルです。PEM 形式の証明書がある場合は、使用する前に PFX に変換する必要があります。

以下の openssl コマンドを使用して PEM 証明書を PFX に変換します。

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

証明書が PFX 形式になったら、証明書をデプロイして **TLSOptions.CertFile** プロパティにそのパスを設定するだけです。

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

ハッシュ証明書

証明書がすでに Windows 証明書ストアにインストールされている場合、証明書のサムプリントを知り、TLSOptions.SChannel_Options プロパティに設定するだけです。

証明書のハッシュを見つけることは、**powershell** で証明書コンテナに対して **dir** コマンドを実行するのと同じくらい簡単です。

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

ハッシュは 16 進数の **Thumbprint** 値です。

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

サムプリント値を取得したら、**TLSOptions.SChannel_Options** プロパティにハッシュと証明書の場所を設定する必要があります。

```
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

クライアントがセキュアサーバーに接続すると、使用されている TLS バージョン (TLS 1.2、TLS 1.3 など)、使用されている暗号、強度などに関する情報をリクエストできます。

この情報にアクセスするには、SChannel Handlerの**GetInfo**関数を呼び出します。SSL Handlerには、SChannel Handlerの作成後に呼び出される**OnSSLAfterCreateHandler**メソッドを使用してアクセスできます。クライアントがサーバーに接続した後、SSL Handlerが割り当てられている場合は、**GetInfo**関数を呼び出します。成功すると、接続データを返します。

```
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, TwsSSLHandler 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 | クライアントがテキストメッセージを送信する

クライアントがサーバーに接続したら、テキストメッセージを送信できます。テキストメッセージを送信するには、WriteData() メソッドを呼び出すだけです。

テキストメッセージの送信

テキストメッセージを送信するには、**WriteData()**メソッドを呼び出します。このメソッドは、呼び出された同じスレッド上で実行されます。

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

QueueOptions.Text が **qmNone** と異なる値の場合、呼び出された同じスレッドで処理される代わりにセカンダリスレッドで処理されます。デフォルトではこのオプションは無効です。

テキストメッセージを送信して応答を待機する

WriteAndWaitData() メソッドを呼び出してテキストメッセージを送信し、サーバーからのレスポンスを待機します。この関数は受信したテキストメッセージを返します。

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

TsgcWebSocketClient | Client Send Binary Message

クライアントがサーバーに接続した後、バイナリメッセージを送信できます。バイナリメッセージを送信するには、WriteData()メソッドを呼び出すだけです。

バイナリメッセージの送信

バイナリメッセージを送信するには、**WriteData()**メソッドを呼び出します。このメソッドは、呼び出された同じスレッド上で実行されます。

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

QueueOptions.Binary が **qmNone** 以外の値の場合、呼び出されたのと同じスレッドではなく、セカンダリスレッドで処理されます。デフォルトではこのオプションは無効です。

バイナリメッセージを送信してレスポンスを待機する

WriteAndWaitData() メソッドを呼び出してバイナリメッセージを送信し、サーバーからのレスポンスを待機します。この関数は受信したバイナリメッセージを返します。

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

TsgcWebSocketClient | クライアントによるテキストおよびバイナリメッセージの送信

WebSocket プロトコルはテキストとバイナリの 2 種類のメッセージのみを許可します。ただし、同じメッセージ内にバイナリデータとテキストを一緒に送信することはできません。これを解決する 1 つの方法は、送信前にバイナリメッセージにヘッダーを追加し、受信時にそのバイナリメッセージをデコードすることです。

sgcWebSocket_Helpers には 2 つの関数があり、バイナリパケットの短い説明を設定するために使用できます。これは基本的にバイナリパケットを識別するためのヘッダーをストリームに追加します。

バイナリメッセージを送信する前に、ストリームをエンコードするためにメソッドを呼び出します。

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

バイナリメッセージを受信したら、ストリームをデコードするためにこのメソッドを呼び出します。

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

唯一の制限は、バイナリメッセージを識別するために使用されるテキストの最大長が10文字であることです（ソースコードにアクセスできる場合は変更可能です）。

TsgcWebSocketClient | テキストメッセージの受信

クライアントがテキストメッセージを受信すると、**OnMessage** イベントが発生します。Text パラメータを読み取って受信したメッセージの文字列を取得します。

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

デフォルトでは、クライアントは **neAsynchronous** メソッドを使用して OnMessage イベントをディスパッチします。これはつまり、このイベントはメインスレッドのコンテキストで実行されるため、例えばフォームのコントロールを更新することがスレッドセーフです。

クライアントが多くのメッセージを受信する場合、または他のスレッドとの同期を制御する必要がある場合は、NotifyEvents プロパティを **neNoSync** に設定します。これにより、OnMessage イベントが**接続スレッドのコンテキストで実行**されます。フォームのコントロールを更新したり共有オブジェクトにアクセスしたりする場合は、独自の同期方法を実装する必要があります。

TsgcWebSocketClient | バイナリメッセージの受信

クライアントがバイナリメッセージを受信すると、**OnBinary** イベントが発火します。受信したバイナリメッセージを取得するには **Data** パラメータを読み取ります。

```
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;
}
```

デフォルトでは、クライアントは **neAsynchronous** メソッドを使用して **OnMessage** イベントをディスパッチします。つまり、このイベントは**メインスレッドのコンテキストで実行される**ため、例えばフォームのコントロールを更新することがスレッドセーフです。

多くのメッセージを受信する場合、または他のスレッドとの同期を制御する必要がある場合は、**NotifyEvents** プロパティを **neNoSync** に設定します。これにより **OnMessage** イベントは**接続スレッドのコンテキストで実行される**ため、フォームのコントロールを更新したり共有オブジェクトにアクセスしたりする場合は、独自の同期メソッドを実装する必要があります。

TsgcWebSocketClient | クライアント認証

TsgcWebSocket クライアントは 4 種類の認証をサポートしています。

- **Basic:** WebSocket ハンドシェイク中に、ユーザーとパスワードを Basic 認証としてエンコードした HTTP ヘッダーを送信します。
- **Token:** WebSocket ハンドシェイク中に HTTP ヘッダーとしてトークンを送信します。サーバーが要求するトークンを `Authentication.Token.AuthToken` に設定してください。
- **Session:** まずクライアントがサーバーに HTTP セッションをリクエストし、サーバーがセッションを返した場合、それは WebSocket ハンドシェイクの GET HTTP ヘッダーで渡されます。 (* sgcWebSockets ライブラリ独自の認証方法)
- **URL :** クライアントは WebSocket ハンドシェイクの GET HTTP ヘッダーを使用して認可をリクエストします (* sgcWebSockets ライブラリ独自の認証方式)。

Basic 認証

これは、ユーザーとパスワードがエンコードされ、HTTPヘッダーとして渡される単純な認可方法です。UserとPasswordを設定し、Basic認可タイプのみを有効にして、この方法を使用してください。

```
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;
```

認証トークン

JWTを使用した認証が可能です。外部ツール（例: HTTP接続、OAuth2など）を使用してトークンを取得する必要があります。

OAuth2コンポーネントをアタッチすると、このトークンを自動的に取得できます。詳しくは[OAuth2](#)をご覧ください。

AuthTokenを設定し、Token Authenticationを有効にする必要があります。

```
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;
```

認証セッション

まずクライアントは HTTP 接続を使用して新しいセッションをリクエストするサーバーに接続します。成功した場合、サーバーは SessionId を返し、クライアントはこの SessionId を WebSocket ハンドシェイクの GET HTTP ヘッダーで送信します。

UserNameとPasswordの設定、およびSession Authenticationの有効化が必要です。

```
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;
```

認証URL

この認証方法は、WebSocketハンドシェイクのGET HTTPヘッダーでユーザー名とパスワードを渡します。

```
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

通信中にエラーが発生することがあります。サーバーは認証されていないか、メッセージの形式が正しくないことを理由に接続を切断する場合があります。エラーがキャプチャされる2つのイベントがあります。

OnError

このイベントは、無効なメッセージタイプ、無効な UTF-8 文字列など、WebSocket プロトコルでエラーが発生するたびに発火します。

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

OnException

このイベントは、アクティブでないソケットへの書き込みや、存在しないオブジェクトへのアクセスなど、例外が発生するたびに発生します。

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

デフォルトでは、サーバーによって接続が閉じられると、例外が発生します。これらの例外を発生させたくない場合は、**Options.RaiseDisconnectExceptions** を無効にしてください。

TsgcWebSocketClient | WebSocket ハンドシェイク

WebSocket プロトコルは HTTP ハンドシェイクを使用して HTTP プロトコルから WebSocket プロトコルにアップグレードします。このハンドシェイクは TsgcWebSocket クライアントコンポーネントによって内部的に処理されますが、サーバーが追加の HTTP ヘッダー情報を必要とする場合は、独自のカスタム HTTP ヘッダーを追加できます。

使用例: HTTP ヘッダー "Client: sgcWebSockets" を追加する必要がある場合

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

TsgcWebSocketClient | クライアント登録プロトコル

デフォルトでは、TsgcWebSocketClient はサブプロトコルを使用しません。WebSocket サブプロトコルは WebSocket プロトコルの上に構築され、カスタムメッセージプロトコルを定義します。WebSocket サブプロトコルの例として MQTT、STOMP などがあります。

WebSocket サブプロトコル名は WebSocket ハンドシェイクの HTTP ヘッダーとして送信されます。このヘッダーはサーバーによって処理され、サーバーがこのサブプロトコルをサポートしていれば接続を受け入れます。サポートされていない場合は接続が自動的に閉じられます。

例: サブプロトコル名'myprotocol'でWebSocketサーバーに接続します

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

TsgcWebSocketClient | クライアントプロキシ

TsgcWebSocket クライアントはプロキシを通じた接続をサポートしています。プロキシ接続を設定するには、TsgcWebSocket クライアントの **Proxy** プロパティを入力してください。

```
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 はプレーン TCP サーバーへの接続を可能にする TCP クライアントコンポーネントです。

はじめに

TsgcTCPClient は、プレーン TCP サーバーへの接続を可能にする TCP クライアントコンポーネントです。TLS/SSL 接続、プロキシサーバー、ハートビートモニタリング、自動再接続 (WatchDog)、および TCP キープアライブをサポートしています。

1. フォームに **TsgcTCPClient** コンポーネントをドロップします。
2. 利用可能な TCP サーバーに接続するために **Host** と **Port** (デフォルトは 80) を設定します。
3. セキュアな接続が必要な場合は **TLS** を **True** に設定します (デフォルトは **False**) 。
4. プロキシサーバーを経由して接続する場合は、オプションで **Proxy** を設定します。
5. イベントを処理します：**OnConnect**、**OnDisconnect**、**OnMessage**、**OnBinary**、**OnError**、**OnException**。
6. 接続を開くには**Active**を**True**に設定します。

リファレンス

- プロパティ — コンポーネントで使用可能な23のプロパティ。
- メソッド — 7 個のパブリックメソッド。
- イベント — 処理できる 12 のイベント。

TsgcTCPClient

TsgcTCPClient はプレーン TCP サーバーへの接続を可能にする TCP クライアントコンポーネントです。

はじめに

TsgcTCPClient は、プレーン TCP サーバーへの接続を可能にする TCP クライアントコンポーネントです。TLS/SSL 接続、プロキシサーバー、ハートビートモニタリング、自動再接続 (WatchDog)、および TCP キープアライブをサポートしています。

1. フォームに **TsgcTCPClient** コンポーネントをドロップします。
2. 利用可能な TCP サーバーに接続するために **Host** と **Port** (デフォルトは 80) を設定します。
3. セキュアな接続が必要な場合は **TLS** を **True** に設定します (デフォルトは **False**) 。
4. プロキシサーバーを経由して接続する場合は、オプションで **Proxy** を設定します。
5. イベントを処理します：**OnConnect**、**OnDisconnect**、**OnMessage**、**OnBinary**、**OnError**、**OnException**。
6. 接続を開くには **Active** を **True** に設定します。

リファレンス

- プロパティ — コンポーネントで使用可能な23のプロパティ。
- メソッド — 7 個のパブリックメソッド。
- イベント — 処理できる 12 のイベント。

TsgcWebSocketServer

TsgcWebSocketServer はサーバー WebSocket コンポーネントを実装し、複数のスレッド化されたクライアント接続を処理できます。

はじめに

TsgcWebSocketServerはサーバーWebSocketコンポーネントを実装しており、複数のスレッド化されたクライアント接続を処理できます。このコンポーネントを構成するには、以下の手順に従ってください。

1. フォームに TsgcWebSocketServer コンポーネントをドロップします。
2. ポートを設定します（デフォルトは 80）。ファイアウォールの内側にある場合は設定が必要になる可能性があります。
3. 許可する仕様を設定します。デフォルトではすべての仕様が許可されています。

RFC6455：標準かつ推奨の WebSocket 仕様です。

Hixie76：これはドラフトであり、Safari 4.2 などの古いブラウザへのサポートを提供したい場合にのみ Hixie76 接続を確立することが推奨されます

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

5. プロシージャを作成して Active = True プロパティを設定します。

リファレンス

- プロパティ — コンポーネントで利用可能な27個のプロパティ。
- メソッド — 7つのパブリックメソッド。
- イベント — 処理可能な22個のイベント。

ガイド

- **Start**
 - [Server Start](#)
 - [Server Bindings](#)
 - [Server Startup - Shutdown](#)
 - [Server Keep Active](#)
- **接続**
 - [サーバー接続の維持](#)
 - [サーバーブレイク TCP](#)
 - [サーバー接続のクローズ](#)
 - [クライアント接続](#)

- **認証**
 - [サーバー認証](#)
- **Send Messages**
 - [Server Send Text Message](#)
 - [Server Send Binary Message](#)
- **メッセージを受信する**
 - [サーバーがテキストメッセージを受信](#)
 - [サーバーがバイナリメッセージを受信](#)
- **SSL**
 - [サーバー SSL](#)
 - [サーバー SSL SChannel](#)

TsgcWebSocketServer | サーバーの起動

サーバーを起動するときに最初に設定する必要があるのはリスニングポートです。デフォルトではポート 80 に設定されていますが、任意のポートに変更できます。
ポートが設定されると、サーバーを起動する 2 つの方法があります。

Active プロパティ

Active プロパティを true に設定すると、サーバーは設定されたポートですべての受信接続のリスンを開始します。

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

Active プロパティを false に設定すると、サーバーは停止し、すべてのアクティブな接続を閉じます。

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

Start / Stop メソッド

Active プロパティを設定するとサーバーが同じスレッドで開始/停止しますが、Start と Stop メソッドはセカンダリスレッドで実行されます。

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

Stop() メソッドを呼び出すと、サーバーは停止し、すべてのアクティブな接続を閉じます。

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

ReStart メソッドを使用して、セカンダリスレッドでサーバーを停止および起動できます。

サーバーを閉じた後にポートを変更して別のポートでリスニングを開始する場合は、前のバインディングを削除するためにサーバーを閉じた後に **Bindings.Clear()** メソッドを呼び出します。そうしないと、サーバーは前のバインディングにバインドしようとします。

TsgcWebSocketServer | Server Bindings

デフォルトでは、**Port** プロパティのみを設定すると、サーバーは**すべての IP** でリスニングポートをバインドします。例えば、127.0.0.1、80.54.11.22、12.55.41.17 の 3 つの IP がある場合、サーバーはこれらすべての IP でポートをバインドします。

必要な IP にのみバインドすることが通常推奨されます。ここで Bindings プロパティを使用できます。Port プロパティを使用する代わりに、Bindings プロパティを使用して 必要な IP と Port を入力するだけです。

例：ポート 5555 を IP 127.0.0.1 および IP 80.58.25.40 にバインドする

```
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;
```

サーバーを閉じた後、異なるポートでリッスンを開始するためにポートを変更する場合は、サーバーを閉じた後に **Bindings.Clear()** メソッドを呼び出して以前のバインディングをすべて削除してください。そうしないと、サーバーは以前のバインディングにバインドしようとします。

TsgcWebSocketServer | サーバーの起動とシャットダウン

サーバーの必要なすべての設定を行った後、サーバーが起動したときと 停止したときを知るために役立つ2つのイベントがあります。

OnStartup

このイベントは、サーバーが起動して新しい接続を処理できるときに発生します。

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

OnShutdown

このイベントは、サーバーが停止し、これ以上接続が受け付けられなくなった後に発生します。

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

TsgcWebSocketServer | サーバーをアクティブに維持する

サーバーが起動すると、何らかの理由で 停止することがあります。予期しないシャットダウン後にサーバーを再起動したい場合は、WatchDogプロパティを使用できます。

WatchDog

WatchDog が有効な場合、サーバーがシャットダウンを検出すると、WatchDog はサーバーが再びアクティブになるまで X 秒ごとに再起動を試みます。

例: 予期しない停止の後、無制限の試行回数で10秒ごとに再起動します。

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


TsgcWebSocketServer | サーバーSSL

サーバーは **SSL証明書** を使用するように設定できます。サーバー証明書を持つ本番サーバーを取得するには、Namecheap、GoDaddy、Thawteなどのよく知られたプロバイダー から証明書を購入する必要があります。テスト目的では **自己署名証明書** を使用できます（自己署名証明書を使用するDemos/Chatの例を確認してください）。

証明書は **PEM 形式** である必要があります。PEM（Privacy Enhanced Mail の略）は RFC 1421～1424 で定義されたコンテナ形式で、パブリック証明書のみ（Apache のインストールや CA 証明書ファイル /etc/ssl/certs など）、またはパブリックキー、プライベートキー、ルート証明書を含む証明書チェーン全体を含む場合があります。単一の PEM 証明書を作成するには、プライベートキーファイルを開き、内容をコピーして証明書ファイルに貼り付けます。

例:

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-----
```

SSLを有効にするには、**SSLプロパティを有効にする** だけで、**CertFile**、**KeyFile**、**RootFile**へのパスを設定します。証明書に証明書チェーン全体（公開鍵、秘密鍵など）が含まれている場合は、すべてのパスを同じ証明書に設定するだけです。

設定する必要がある別のプロパティは **SSLOptions.Port** で、これはセキュア接続に使用されるポートです。

シンプルな SSL 設定

例: IP 127.0.0.1 とポート 443 で SSL を設定します

```
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および非SSL

サーバーを複数の IP とポートでリッスンするよう設定できます。動作の詳細については[バインディングの記事](#)を参照してください。サーバーは SSL 接続と非 SSL 接続を同時に受け付けるように設定できます（もちろん、異なるポートでリッスンします）。2つの異なるポートにバインドし、一方のポートを SSL 接続用、もう一方を非 SSL 接続用に設定するだけです。

例: サーバーを IP 127.0.0.1、ポート 80（非暗号化）および 443（SSL）で設定する

```
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;
```

サーバー SSL | Indy サーバー向け SChannel

Indy ベースのサーバーコンポーネント (**TsgcWebSocketServer**、**TsgcWebSocketHTTPServer**) は、OpenSSL の代わりに **Windows SChannel** (Secure Channel) を TLS プロバイダーとして使用できます。SChannel はネイティブの Windows TLS 実装であるため、外部 DLL は不要です。

動作の仕組み

サーバーコンポーネントで SSL が有効になっており、IOHandler が **iohSChannel** に設定されている場合、サーバーは Windows SChannel API を使用してすべての TLS 操作を処理する **TsgcIdServerIOHandlerSSLChannel** インスタンスを作成します。受信クライアント接続ごとに、サーバーは SChannel プロバイダーを通じて TLS ハンドシェイクを実行し、プロトコルバージョン、暗号スイートをネゴシエートし、設定された証明書をバインドします。

SChannel は**Windows 証明書ストア**または **PFX ファイル** (.pfx / .p12) から証明書を読み込みます。PEM ファイルは不要で、OpenSSL ライブラリをデプロイする必要もありません。

設定

Indy サーバーで SChannel を有効にするには、以下のプロパティを設定してください。

1. **SSL** プロパティを **True** に設定します。
2. **SSLOptions.IOHandler** を **iohSChannel** に設定します。
3. **SSLOptions.Version** を 目的の TLS バージョン (tls1_2、tls1_3 など) に設定します。
4. **SSLOptions.Port** を セキュア接続に使用するポートに設定します。
5. 以下に説明する 2 つの方法のいずれかを使用して 証明書を設定します。

SChannel_Options プロパティ

SSLOptions.SChannel_Options サブプロパティは SChannel 固有の設定を公開します:

プロパティ	説明
CertHash	Windows 証明書ストアにインストールされた証明書のサムプリント (16 進数のハッシュ)。
CertStoreName	検索する証明書ストア: scsnMY (個人)、 scsnRoot 、 scsnTrust 、 scsnCA 。
CertStorePath	ストアの場所: scspStoreLocalMachine または scspStoreCurrentUser です。
CipherList	オプションのコロン区切り暗号アルゴリズムリスト (例: CALG_AES_256:CALG_AES_128)。空の場合はシステムのデフォルトを使用します。
UseLegacyCredentials	True の場合、新しい SCH_CREDENTIALS 構造の代わりにレガシーの SCHANNEL_CRED 構造を使用します。新しい API をサポートしていない古い Windows バージョンでこれを有効にしてください。

さらに、一般的な **SSLOptions** プロパティの **CertFile** と **Password** は、PFX ファイルから証明書を読み込む際に使用されます。

Windows ストアからの証明書

証明書がすでに Windows 証明書ストアにインストールされている場合は、証明書のサムプリントを提供し、それがどこにあるかを指定してください。

サムプリントを確認するには、**PowerShell** を開いて実行してください:

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

Thumbprint 列に 必要な16進数ハッシュが表示されます。

```
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 ファイルからの証明書

PFX (.pfx または .p12) 証明書ファイルをお持ちの場合は、SSLOptionsで**CertFile** と **Password** プロパティを設定します。SChannelは起動時に証明書をインポートします。

```
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 証明書と秘密鍵をお持ちの場合は、まず OpenSSL を使用して PFX 形式に変換してください:

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

TLS バージョン

SSLOptions.Version プロパティを使用して、サーバーが受け付ける TLS バージョンを制御します:

値	説明
tls1_0	TLS 1.0 (非推奨)
tls1_1	TLS 1.1
tls1_2	TLS 1.2 (推奨)
tls1_3	TLS 1.3
tlsUndefined	TLS 1.0、1.1、1.2を受け入れます

暗号スイートリスト

デフォルトでは、SChannelはシステムの暗号構成を使用します。**SChannel_Options.CipherList**を コロン区切りのアルゴリズム名のリストに設定することで、許可される暗号を制限できます。例:

```
CALG_AES_256:CALG_AES_128
```

Windowsのデフォルトを使用するには、このプロパティを空のままにしてください。

レガシー資格情報

Windows Server 2019以前では、新しい **SCH_CREDENTIALS** APIがサポートされない場合があります。古い Windowsバージョンでサーバーの起動に失敗した場合は、レガシーの **SCHANNEL_CRED** 構造を使用するために **SChannel_Options.UseLegacyCredentials** を **True** に設定してください。

コンポーネントはほとんどの場合Windowsバージョンを自動的に検出しますが、必要に応じてレガシーモードを強制することもできます。

OpenSSL に対する利点

- **外部 DLL 不要**： SChannel は Windows に組み込まれているため、libeay32.dll / ssleay32.dll または libcrypto / libssl を配備する必要はありません。
- **Windows証明書ストアの統合**： オペレーティングシステムによってすでにインストールおよび管理されている証明書を使用します。
- **自動更新**： TLS の改善とセキュリティパッチはWindows Updateを通じて適用されます。

注意

- SChannel は**Windows のみ**で利用可能です。クロスプラットフォームサーバーには OpenSSL (iohOpenSSL) を使用してください。
- サーバーは証明書に関連付けられた**秘密鍵**を持っている必要があります。Windows ストアの方法を使用する場合、証明書は秘密鍵とともにインポートされている必要があります。
- Certificate Store方式を使用する本番サーバーでは、サービスを実行するユーザーアカウントに関係なく証明書が利用可能になるように、**Local Machine**ストア (scspStoreLocalMachine) の使用をお勧めします。
- サポートされているのは**PFX** (.pfx / .p12) 証明書 ファイルのみです。PEMファイルがある場合は、まず PFX形式に 変換してください。

TsgcWebSocketServer | Server Verify Certificate

デフォルトでは、サーバーはピア証明書を検証しません。クライアント証明書を検証するようにサーバーを設定するには、次の手順を実行してください：

1. `SSLOptions.VerifyCertificate = true` を設定してください。

`OnSSLVerifyPeer` イベントを処理し、クライアントが証明書で接続するたびに通知を受けるには、次のコードを実装してください。

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

イベント `OnSSLVerifyPeer` はクライアントが証明書を提供した場合にのみ呼び出されます。クライアントが証明書を提供しない場合、このイベントは発火しません。

証明書を使用する **SSL 接続のみ**を許可するようにサーバーを設定できます。これを行うには、次のプロパティを設定してください：

- `SSLOptions.VerifyCertificate_Options.FailIfNoCertificate = true`

クライアントが証明書を提供しない場合、SSL ハンドシェイクで接続が閉じられます。

TsgcWebSocketServer | サーバーの接続維持

クライアントがサーバーに接続した後、電波状況の悪化、接続エラーなどにより接続が切断されることがあります。HeartBeat を使用して接続を維持します。

HeartBeat

HeartBeat プロパティを使用すると、X 秒ごとに **Ping** を送信して**接続を維持**できます。一部のクライアントは、ピア間でデータが交換されない場合に TCP 接続を閉じます。HeartBeat は、特定の間隔で ping を送信することでこの問題を解決します。通常、これは接続を維持するのに十分ですが、X 秒後にクライアントからのレスポンスが受信されなかった場合に接続を閉じるには、TimeOut 間隔を設定できます。

使用例： 30 秒ごとにすべての接続クライアントに ping を送信する。

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

TsgcWebSocketServer | サーバープレーン TCP

WebSocketサーバーは、WebSocket、HTTP、SSE、およびその他のプロトコルを受け付けますが、プレーンな TCP 接続でも動作できます。詳しくは[TCP Connections](#)をご覧ください。

TCP 接続を処理するために使用できるイベントが2つあります。

OnTCPConnect

このイベントはクライアントがサーバーに接続した後、クライアントとサーバー間のハンドシェイクの前に呼び出されます。OnConnect イベントはクライアントがメッセージを送信した後にのみ発生します (サーバーが使用されているプロトコルを検出できるようにするため)。

このイベントを使用すると、新しいクライアントがサーバーに接続しようとしていることを把握できます。サーバーは接続を受け入れるか拒否できます。デフォルトでは、サーバーは常に接続を受け入れます。

OnUnknownProtocol

このイベントは、サーバーがクライアントから最初のメッセージを受信したが、既知のプロトコルのいずれかであるかどうかを検出できない場合に呼び出されます。このイベントでは、サーバーはプロトコルを受け入れるか拒否するかを決定できます。

OnConnect

このイベントは接続が成功し完全に確立された後に発生します。接続がプレーン TCP の場合、OnUnknownProtocol イベントでプロトコルが受け入れられた後に発生します。

TsgcWebSocketServer | サーバー接続のクローズ

単一のConnectionは、CloseまたはDisconnectメソッドを使用して閉じることができます。

切断

[TsgcWSConnection](#) には `Disconnect()` というメソッドがあり、ソケットレベルで接続を切断できます。このメソッドを呼び出すと、クライアントからの応答を待たずに直接ソケットが切断されます。このメソッドでクローズコードを送信できます。

閉じる

[TsgcWSConnection](#) には `Close()` メソッドがあります。これにより、接続を閉じるよう要求するメッセージをクライアントに送信できます。クライアントがこのメッセージを受信すると、接続を閉じる必要があり、サーバーは接続が閉じられたという通知を受け取ります。このメソッドでクローズコードを送信できます。

DisconnectAll

すべてのアクティブな接続を切断します。このメソッドはサーバーがリスンを停止する前に自動的に呼び出されますが、いつでも呼び出すことができます。

TsgcWebSocketServer | クライアント接続

アクティブなクライアント接続にアクセスするには、Connections プロパティを使用してリストを反復処理し、クライアント接続クラスにアクセスします。Connections プロパティはスレッド化されたリストにアクセスするため、まずリストをロックし、完了したらロックを解除してください。

```
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 | サーバー認証

TsgcWebSocket サーバーは 3 種類の認証をサポートしています:

- **Basic**: WebSocket ハンドシェイク中に、Basic 認証としてエンコードされたユーザーとパスワードを含む HTTP ヘッダーを読み取ります。
- **Session**: クライアントはまずサーバーに HTTP セッションをリクエストし、サーバーがセッションを返すと、WebSocket ハンドシェイクの GET HTTP ヘッダーに渡されます。 (* sgcWebSockets ライブラリ独自の認証方法)。
- **URL**: WebSocket ハンドシェイクの GET HTTP ヘッダーを使用してリクエストの認証を読み取ります。 (* sgcWebSockets ライブラリ独自の認証方法)。

AuthUsers プロパティを使用して、認証済みユーザーのリストを設定できます。次の形式でユーザーを設定するだけです: user=password

OnAuthentication

サーバーがクライアントから認証リクエストを受信するたびに、このイベントが呼び出されてユーザーが認証されているかどうかを示します。

接続を受け入れるかどうかを指定するには **Authenticated** パラメータを使用します。

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

OnUnknownAuthentication

JWT など、デフォルトでサポートされていない認証タイプの場合でも、このイベントを使用して接続を受け入れるか拒否するかを決定できます。パラメーターを読み取り、接続を受け入れるかどうかを判断してください。

```
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 | サーバーテキストメッセージ送信

クライアントがサーバーに接続すると、サーバーはテキストメッセージを送信できます。テキストメッセージを送信するには、WriteData() メソッドを呼び出して単一のクライアントにメッセージを送信するか、Broadcast を使用してすべてのクライアントにメッセージを送信してください。

テキストメッセージの送信

WriteData() メソッドを呼び出してテキストメッセージを送信します。

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

QueueOptions.Text が **qmNone** と異なる値の場合、呼び出された同じスレッドで処理される代わりにセカンダリスレッドで処理されます。デフォルトではこのオプションは無効です。

プロパティ **IOHandlerOptions.IOHandlerType = iohIOCP** の場合、**QueueOptions** は機能しません (IOCP のアーキテクチャ上、この機能はサポートされていません)。

TsgcWSConnection から WriteData() メソッドを呼び出すこともできます。例：クライアントがサーバーに接続したときにメッセージを送信します。

```
void OnConnect(TsgcWSConnection *Connection)
{
    Connection->WriteData("Hello From Server");
}
```

接続されているすべてのクライアントにメッセージを送信

Broadcast() メソッドを呼び出して、接続しているすべてのクライアントにテキストメッセージを送信します。

```
TsgcWebSocketServer1->Broadcast("Hello From Server");
```

TsgcWebSocketServer | サーバーバイナリメッセージ送信

クライアントがサーバーに接続すると、サーバーはバイナリメッセージを送信できます。バイナリメッセージを送信するには、WriteData() メソッドを呼び出して単一のクライアントにメッセージを送信するか、Broadcast を使用してすべてのクライアントにメッセージを送信します。

バイナリメッセージの送信

WriteData() メソッドを呼び出してバイナリメッセージを送信します。

```
TsgcWebSocketServer1->WriteData("guid", new TMemoryStream());
```

QueueOptions.Binary が qmNone と異なる値の場合、呼び出されたのと同じスレッドで処理されるのではなく、セカンダリスレッドで処理されます。デフォルトではこのオプションは無効です。

QueueOptions は、プロパティ IOHandlerOptions.IOHandlerType = iohIOCP の場合には機能しません（IOCP アーキテクチャのため、この機能はサポートされていません）。

TsgcWSConnection から WriteData() メソッドを呼び出すこともできます。例：クライアントがサーバーに接続したときにメッセージを送信します。

```
void OnConnect(TsgcWSConnection *Connection)
{
    Connection->WriteData(new TMemoryStream());
}
```

接続されているすべてのクライアントにメッセージを送信

Broadcast() メソッドを呼び出して、接続されているすべてのクライアントにバイナリメッセージを送信します。

```
TsgcWebSocketServer1->Broadcast(new TMemoryStream());
```

TsgcWebSocketServer | Server Receive Text Message

サーバーがテキストメッセージを受信すると、**OnMessage** イベントが発生します。Text パラメータを読み取って、受信したメッセージの文字列を取得します。

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

デフォルトでは、サーバーは OnMessage イベントをディスパッチするために **neAsynchronous** メソッドを使用します。これはこのイベントがメインスレッドのコンテキストで実行されることを意味し、フォームのコントロールを更新する際にスレッドセーフです。

サーバーが大量のメッセージを受信する場合、または他のスレッドとの同期を制御する必要がある場合は、NotifyEvents プロパティを **neNoSync** に設定します。これは、OnMessage イベントが接続スレッドのコンテキストで実行されることを意味します。そのため、フォームのコントロールを更新したり、共有オブジェクトにアクセスしたりする必要がある場合は、独自の同期 メソッドを実装する必要があります。

TsgcWebSocketServer | サーバーがバイナリメッセージを受信する

サーバーがバイナリメッセージを受信すると、**OnBinary** イベントが発生します。受信したバイナリメッセージを取得するには Data パラメータを読み取ってください。

```
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();
}
```

デフォルトでは、サーバーは OnMessage イベントをディスパッチするために **neAsynchronous** メソッドを使用します。これはこのイベントがメインスレッドのコンテキストで実行されることを意味し、フォームのコントロールを更新する際にスレッドセーフです。

サーバーが大量のメッセージを受信する場合、または他のスレッドとの同期を制御する必要がある場合は、NotifyEventsプロパティを**neNoSync**に設定します。これは、OnMessage イベントが接続スレッドのコンテキストで実行されることを意味します。そのため、フォームのコントロールを更新したり、共有オブジェクトにアクセスしたりする必要がある場合は、独自の同期 メソッドを実装する必要があります。

TsgcWebSocketServer | サーバーがクライアントからヘッダーを読み取る

クライアントが接続すると、クライアント接続に関する情報を含むヘッダーのリストが WebSocket サーバーに送信されます。これらのクライアントヘッダーを読み取るには、サーバーコンポーネントの **OnHandshake** イベントを使用できます。このイベントはサーバーがクライアントからヘッダーを受信してレスポンスを送信する前に呼び出されます。

クライアントヘッダーは **TsgcWSConnectionServer** の **HeadersRequest** プロパティに格納されます。

```
void OnServerHandshake(TsgcWSConnection *Connection, TStringList *Headers)
{
    ShowMessage(dynamic_cast<TsgcWSConnectionServer*>(Connection)->HeadersRequest->Text);
}
```


TsgcWebSocketHTTPServer

AResponseInfo.

はじめに

TsgcWebSocketHTTPServer はサーバー WebSocket コンポーネントを実装し、[TsgcWebSocketServer](#) と同様に複数のスレッド化されたクライアント接続を処理でき、WebSocket 接続と HTTP リクエストを同じポートで共有しながら、組み込みの HTTP サーバーを使用して HTML ページを提供できます。

このコンポーネントの設定手順に従います:

1. フォームに TsgcWebSocketHTTPServer コンポーネントをドロップします。
2. ポートを設定します (デフォルトは 80)。ファイアウォールの内側にある場合は設定が必要になる可能性があります。
3. 許可する仕様を設定します。デフォルトではすべての仕様が許可されています。

RFC6455: 標準かつ推奨の WebSocket 仕様です。

Hixie76: これはドラフトであり、Safari 4.2 などの古いブラウザへのサポートを提供したい場合にのみ Hixie76 接続を確立することが推奨されます

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

OnHTTPUploadBeforeCreatePostStream: このイベントはヘッダーが読み取られた後、POSTストリームが作成される前に呼び出されます。

* 場合によっては、不審な接続によってCPUの使用率が高くなることがあります。この場合、HTTPリクエストであればエラー500を返すか、不明なプロトコルリクエストの場合は接続を閉じてください。

5. プロシージャを作成し、プロパティ Active = true を設定します。

リファレンス

- プロパティ — コンポーネントで利用可能な33個のプロパティ。
- メソッド — 9つのパブリックメソッド。
- イベント — 処理できる36のイベント。

ガイド

- HTTP
 - [HTTP サーバーリクエスト](#)
 - [HTTP ファイルのディスパッチ](#)
 - [HTTP/2 サーバー](#)
 - [HTTP/2 サーバープッシュ](#)
 - [HTTP/2 代替サービス](#)
 - [HTTP/2 サーバースレッド](#)
 - [HTTP 大きなファイルの POST](#)
 - [レスポンスボディなしの HTTP 404 エラー](#)

- **SSL**
 - [Server SSL SChannel](#)
- **その他**
 - [HTTP Server Sessions](#)

TsgcWebSocketHTTPServer | HTTP Server Requests

OnCommandGet を使用して HTTP クライアントリクエストを処理します。以下のパラメータを使用します。

- **RequestInfo**: HTTP リクエスト情報を含みます。
- **ResponseInfo**: HTTP リクエストに対する HTTP レスポンスです。
 - **ContentText**: テキスト形式のレスポンスです。
 - **ContentType**: Content-Type の種類です。
 - **ResponseNo**: HTTP レスポンスの番号。例: 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

HTTP レスポンスをカスタマイズするには、このイベントを使用します。たとえば、一部のエンドポイントには認可スキームを使用させ、他のエンドポイントには認可なしでアクセスさせたい場合は、options パラメータを使用してそれを許可または無効にします。以下は、Authorization Basic が有効になっているが、ユーザーがエンドポイント/public をリクエストすると認可が不要になる例です。

```
void __fastcall TForm1::OnBeforeCommand(TsgcWSConnection* aConnection, TIdHTTPRequestInfo* ARequestInfo, TIdHTTPF
{
  if (ARequestInfo->Document == "/public")
    aOptions << hcoAuthorizedBasic;
}
```

TsgcWebSocketHTTPServer | HTTP Dispatch Files

クライアントがファイルをリクエストすると、**OnCommandGet** イベントが発生しますが、**DocumentRoot** プロパティを使用してファイルを自動的にディスパッチできます。

例: **DocumentRoot** を **c:/www/files** に設定した場合、新しいファイルがリクエストされるたびにサーバーはこのフォルダーを検索します。ファイルが存在する場合、自動的に配信されます。

TsgcWebSocketHTTPServer | HTTP/2 サーバー

sgcWebSockets HTTP サーバーは HTTP/1.1 および HTTP/2.0 リクエストを処理できます。サーバーの HTTP2Options を使用して HTTP/2 プロトコルを有効にできます。

HTTP2Options.Enabled = true に設定してサーバーが HTTP/2 プロトコルリクエストを受け付けられるようにします。リクエストは HTTP/1.1 プロトコルと同様にユーザーが処理できます。[詳細を読む](#)。

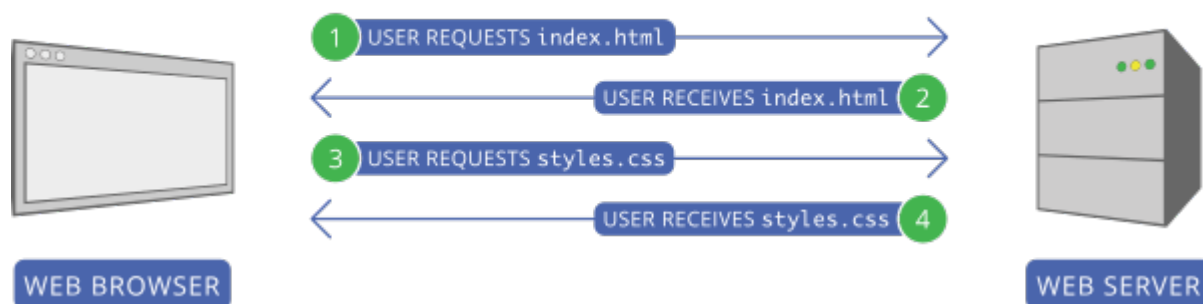
HTTP/2 プロトコルが有効な場合でも、サーバーは引き続き HTTP/1.1 リクエストをサポートします。

デフォルトでは、新しいHTTP/2接続があっても OnConnect および OnDisconnect イベントは呼び出されませんが、HTTP2Options.Events プロパティにアクセスすることでこれを変更できます。新しいHTTP/2接続や切断のたびに通知を受け取るかどうかをカスタマイズできます。

TsgcWebSocketHTTPServer | HTTP/2 Server Push

HTTP は通常、リクエスト/レスポンスパターンで動作します。クライアントがサーバーにリソースの REQUEST を送信し、サーバーがリクエストされたリソースまたはエラーを含む RESPONSE を返します。通常、ブラウザなどのクライアントはサーバーが提供するアセットに対して一連のリクエストを行います。

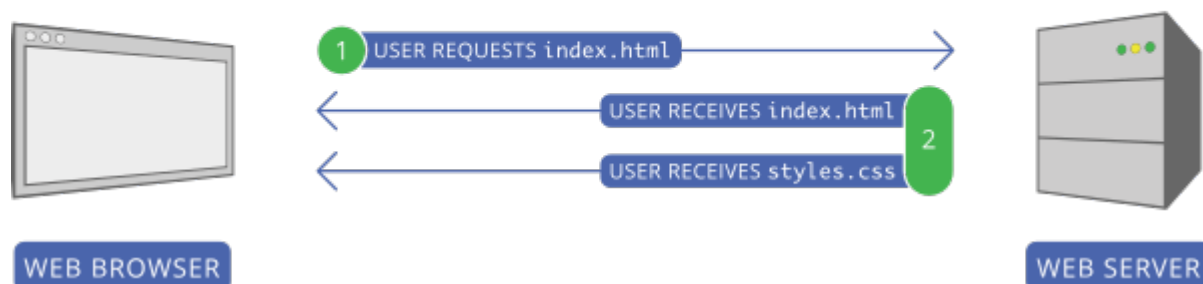
TYPICAL WEB SERVER COMMUNICATION



このアプローチの主な問題は、クライアントがまずリソース(たとえばindex.html)を取得するリクエストを送信し、サーバーがレスポンスを送信するまで待機し、コンテンツを読み取り、その後他のすべてのリクエスト(たとえばstyles.css)を行わなければならないことです。

HTTP/2 サーバープッシュはこの問題を解決しようとしています。クライアントがファイルをリクエストすると、サーバーがそのファイルに追加のファイルが必要と判断した場合、それらのファイルは自動的にクライアントにプッシュされます。

WEB SERVER COMMUNICATION WITH HTTP/2 SERVER PUSH



上記のスクリーンショットでは、クライアントが最初にindex.htmlをリクエストしています。サーバーはこのリクエストを読み取り、レスポンスとして2つのファイル (index.htmlとstyles.css) を送信します。これにより、styles.cssを取得するための2回目のリクエストが不要になります。

サーバープッシュの設定

上記のスクリーンショットに従って、/index.html ファイルへの新しいリクエストがあるたびに、サーバーがindex.html と styles.css を送信するようにサーバーを設定できます。

各リクエストをプッシュプロミスリストと関連付けるために **PushPromiseAddPreLoadLinks** メソッドを使用してください。

```
TsgcWebSocketHTTPServer *server = new TsgcWebSocketHTTPServer(this);
```

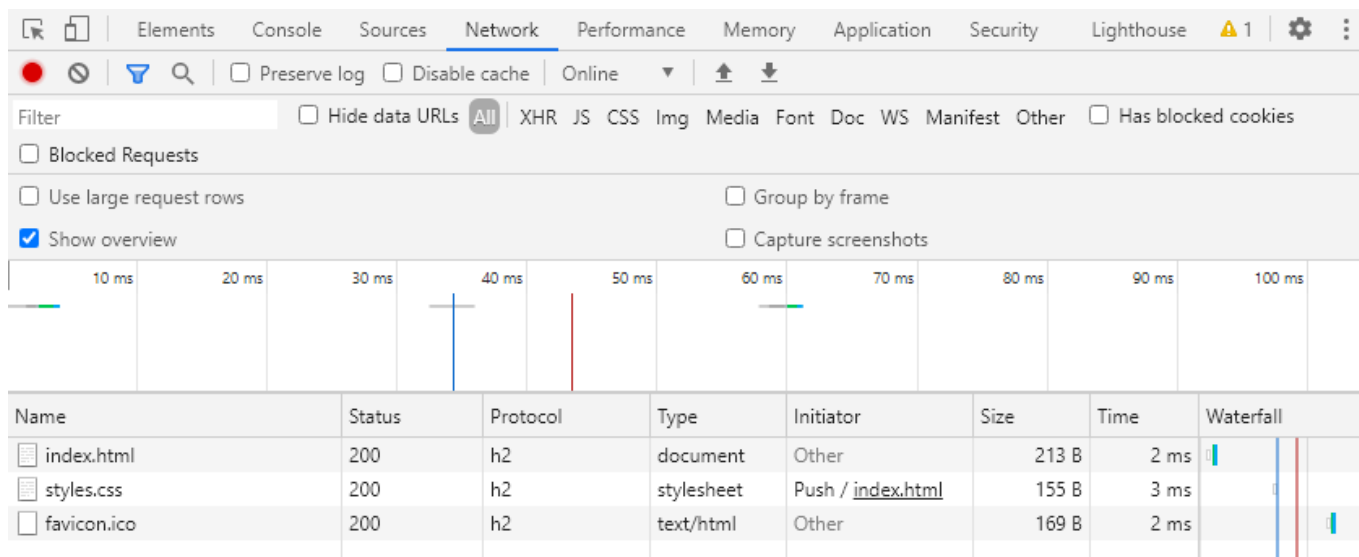
```

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デベロッパーツールを使用すると、styles.cssファイルがどのようにクライアントへプッシュされるかを確認できます。



TsgcWebSocketHTTPServer | HTTP/2 代替サービス

Alt-Svc HTTPヘッダーは、同じリソースを別のサービスやプロトコルから到達できることをクライアントに通知するために使用されます。たとえば、サーバーがHTTP/2をサポートしていることをHTTPクライアントに通知する場合に役立ちます。

例：サーバーがローカル IP 127.0.0.1 で稼働していて、2つのポートでリッスンしている場合：80（非暗号化）と443（暗号化）。以下のHTTPヘッダーを使用して、ポート443でHTTP/2がサポートされていることをクライアントに通知できます

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

HTTP/2が有効な場合、接続がHTTP/2プロトコルで動作していないとこのヘッダーが自動的に追加されます。プロパティ **HTTP2Options.AltSvc** を使用してこの機能を有効または無効にできます。

TsgcWebSocketHTTPServer | HTTP/2 サーバースレッド

HTTP 1.1 と HTTP 2.0 の違いを以下に示します:

HTTP 1.1

従来の HTTP の動作では、同一接続で複数のリクエストを行う場合、クライアントは次を送信する前に各リクエストのレスポンスを待つ必要がありました。この逐次的なアプローチはウェブサイトのリソースのロード時間を大幅に増加させます。この問題に対処するため、HTTP/1.1 ではパイプラインと呼ばれる機能が導入され、クライアントはサーバーのレスポンスを待たずに複数のリクエストを送信できるようになりました。サーバーはクライアントに受信した順序でレスポンスします。

パイプライン化は解決策のように見えてましたが、いくつかの課題がありました:

- **サーバーの無視またはレスポンスの破損:** 一部のサーバーはパイプラインされたリクエストを無視するかレスポンスを破損し、信頼性の低い通信につながりました。
- **Head-of-Line Blocking:** パイプライン内の最初のリクエストが後続のリクエストをブロックし、他のリクエストの処理に遅延を引き起こす可能性があります。head-of-line blockingとして知られるこの現象により、ページの読み込み時間が遅くなりました。

HTTP/1.1 をサポートするサーバーからのページロードを最適化するために、ウェブブラウザは回避策を実装しました。サーバーへの6~8本の並列接続を開き、複数のリクエストの同時送信を可能にします。この並列性はパイプラインに関連する問題を軽減し、全体的なページロード時間を改善することを目的としています。

ウェブブラウザが 6 から 8 の並列接続を選択する理由は最適化の考慮に基づいています。この数値を選択した具体的な理由には、リソース使用率、ネットワーク効率、および潜在的なボトルネックの回避のトレードオフが含まれる場合があります。

HTTP 2.0

パイプライン処理の制約に対応するため、HTTP/2 はマルチプレキシングと呼ばれる機能を導入しました。**マルチプレキシング**は、**単一の接続上で複数のリクエストとレスポンスを同時に送信**することを可能にし、クライアントとサーバー間のより**効率的な通信**を実現します。

HTTP/2 はバイナリフレーミングメカニズムを使用しています。これは HTTP メッセージがフレームと呼ばれる小さな独立した単位に分解されることを意味します。これらのフレームはインターリーブでき、互いに独立して接続上で送信できます。受信側では、フレームが再組み立てされて元の HTTP メッセージが再構成されます。

このバイナリフレーミングメカニズムは HTTP/2 での多重化を実現するための基本要素です。ブラウザが同じ接続で複数のリクエストをブロッキングなしに送信できるようにします。その結果、Chrome などのブラウザは HTTP/2 リクエストに同じ接続 ID を使用し、クライアントとサーバー間の効率的で途切れのない通信を実現します。本質的に、バイナリフレーミングメカニズムによって実現された HTTP/2 の多重化機能は、単一の接続を通じた複数のリクエストとレスポンスの同時伝送を促進することで、クライアントとサーバー間のデータ交換の効率と速度を向上させます。

TsgcWebSocketHTTPServer

HTTP/2 プロトコルのパフォーマンスを向上させるために、サーバーが新しい HTTP/2 リクエストを受信するたびに、リクエストはデフォルトでスレッドプール (デフォルトで 32) でディスパッチされます。これにより、単一の

接続が多数の並行リクエストを送信し、このスレッドプールがない場合は (接続スレッドのコンテキストで) 順次処理が必要になるような状況での待機を回避します。

スレッドプールの動作は、以下のプロパティで設定できます。

- **HTTP2Options.PoolOfThreads.Enabled:** (デフォルトはfalse) これを有効にすると、HTTP/2要求を接続スレッドの代わりにスレッドプールでディスパッチします。
- **HTTP2Options.Threads:** (デフォルトは32) HTTP/2リクエストを処理するスレッド数。サーバーのプロセッサ数に応じて数を設定します。

リクエストを微調整して、時間のかかるリクエストをスレッドプールで処理し、その他は接続スレッドで処理するように選択するには、**OnHttp2BeforeAsyncRequest** イベントを使用します。このイベントはスレッドプールにリクエストをキューイングする前に発生します。**Async** パラメータを使用してリクエストをスレッド化するかどうかを設定します。

```
void __fastcall TForm1::OnHTTP2BeforeAsyncRequest(TObject *Sender, TsgcWSConnection *Connection, const TIdHTTPRec
{
    if (ARequestInfo.Document == "/fast-request")
        Async = false;
}
```

TsgcWebSocketHTTPServer | レスponse本文のない404エラー

デフォルトでは、Indy ライブラリは `ContentText` または `ContentStream` が割り当てられていない場合に HTTP レスponseにコンテンツボディを追加します。空のレスponseボディを返したい場合（404 エラーなど）は、以下のアプローチを使用できます。

コンテンツのない新しい `TStringStream` を作成し、HTTP レスponseの `ContentStream` プロパティに割り当てます。これにより、HTTP レスponseはデフォルトの HTML タグなしで送信されます。

使用例

```
private void OnCommandGet(TIdContext *AContext, TIdHTTPRequestInfo *ARequestInfo,
    TIdHTTPResponseInfo *AResponseInfo)
{
    AResponseInfo->ContentStream = new TStringStream("");
    AResponseInfo->ContentType = "text/html";
    AResponseInfo->ResponseNo = 404;
}
```

TsgcWebSocketHTTPServer | セッション

HTTPはステートレスプロトコルです（少なくともHTTP 1.1まで）。クライアントはファイルをリクエストし、サーバーはレスポンスを送信し、接続が閉じられます（keep-aliveを有効にすることで接続が即座に閉じられないようにできますが、これはこの記事の範囲外です）。セッションを使用すると、クライアントに関する情報を保存できます。例えば、クライアントのログイン時に使用できます。任意の一意のセッションIDを使用し、セッションリストを検索して既存のセッションがあるかどうかを確認し、なければ新しいセッションを作成できます。セッションは一定期間の非アクティブ後に破棄するか、クライアントのログアウト後に手動で破棄できます。

設定

TsgcWebSocketHTTPServerには、サーバーコンポーネントでセッションを有効/無効にするいくつかのプロパティがあります。最も重要なものは次のとおりです。

プロパティ	説明
SessionState	Tこれはセッションを使用するために最初に有効にする必要があるプロパティです。このプロパティが有効になっていない場合、セッションは機能しません。
SessionTimeout	ここでは、セッションがアクティブになる最大時間（ミリ秒）としてゼロより大きい値を設定する必要があります。
AutoStartSession	セッションは自動的に作成（AutoStartSession = true）または手動で作成（AutoStartSession = false）できます。セッションが自動的に作成される場合、サーバーはアクティブなセッションを持っているかどうかを確認するために RemoteIP を一意識別子として使用します。

```
TsgcWebSocketHTTPServer1->SessionState = true;
TsgcWebSocketHTTPServer1->SessionTimeout = 600000;
AutoStartSession = false;
```

セッションを作成します

新しいセッションを作成するには、一意の新しいセッション ID を作成する必要があります。任意の値を使用できます。**使用例:** クライアントが認証する場合、セッション ID としてユーザー名 + パスワード + リモート IP を使用できます。

次に、セッションリストを検索してすでに存在するか確認します。存在しない場合は新しいセッションを作成します。

新しいセッションが作成されると **OnSessionStart** イベントが呼び出され、セッションが閉じられると **OnSessionEnd** イベントが発生します。

```
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 | ビデオのストリーミング

サーバーを使用してビデオファイルをストリーミングする場合、IndyStreamFileVideo 関数を使用してチャンク転送エンコーディングでファイルをストリーミングできます。

この関数は以下のパラメータを受け取ります。

- **AContext:** OnCommandGet イベントから取得します。
- **AResponseInfo :** OnCommandGet イベントから取得されます。
- **aFileName:** ストリーミングするビデオの完全なファイル名です。
- **ContentType:** デフォルトは「video/mpeg」です。
- **aBufferSize:** デフォルトは 1024 バイトです。

使用例: ユーザーがドキュメント /video.mp4 にアクセスしたとき、このビデオが c:\videos\video.mp4 フォルダにある場合にビデオをストリーミングします

```
void OnServerCommandGet(TIdContext *AContext, TIdHTTPRequestInfo *ARequestInfo, TIdHTTPResponseInfo *AResponseInfo)
{
    if (ARequestInfo->Document == "/video.mp4") {
        IndyStreamFileVideo(AContext, AResponseInfo, "C:\\videos\\video.mp4");
    } else {
        AResponseInfo->ResponseNo = 404;
    }
}
```

サーバー SSL | Indy サーバー向け SChannel

Indy ベースのサーバーコンポーネント (**TsgcWebSocketServer**、**TsgcWebSocketHTTPServer**) は、OpenSSL の代わりに **Windows SChannel** (Secure Channel) を TLS プロバイダーとして使用できます。SChannel はネイティブの Windows TLS 実装であるため、外部 DLL は不要です。

動作の仕組み

サーバーコンポーネントで SSL が有効になっており、IOHandler が **iohSChannel** に設定されている場合、サーバーは Windows SChannel API を使用してすべての TLS 操作を処理する **TsgcIdServerIOHandlerSSLSChannel** インスタンスを作成します。受信クライアント接続ごとに、サーバーは SChannel プロバイダーを通じて TLS ハンドシェイクを実行し、プロトコルバージョン、暗号スイートをネゴシエートし、設定された証明書をバインドします。

SChannel は**Windows 証明書ストア**または **PFX ファイル** (.pfx / .p12) から証明書を読み込みます。PEM ファイルは不要で、OpenSSL ライブラリをデプロイする必要ありません。

設定

Indy サーバーで SChannel を有効にするには、以下のプロパティを設定してください。

1. **SSL** プロパティを **True** に設定します。
2. **SSLOptions.IOHandler** を **iohSChannel** に設定します。
3. **SSLOptions.Version** を 目的の TLS バージョン (tls1_2、tls1_3 など) に設定します。
4. **SSLOptions.Port** を セキュア接続に使用するポートに設定します。
5. 以下に説明する 2 つの方法のいずれかを使用して 証明書を設定します。

SChannel_Options プロパティ

SSLOptions.SChannel_Options サブプロパティは SChannel 固有の設定を公開します:

プロパティ	説明
CertHash	Windows 証明書ストアにインストールされた証明書のサムプリント (16 進数のハッシュ)。
CertStoreName	検索する証明書ストア: scsnMY (個人)、 scsnRoot 、 scsnTrust 、 scsnCA 。
CertStorePath	ストアの場所: scspStoreLocalMachine または scspStoreCurrentUser です。
CipherList	オプションのコロン区切り暗号アルゴリズムリスト (例: CALG_AES_256:CALG_AES_128)。空の場合はシステムのデフォルトを使用します。
UseLegacyCredentials	True の場合、新しい SCH_CREDENTIALS 構造の代わりにレガシーの SCHANNEL_CRED 構造を使用します。新しい API をサポートしていない古い Windows バージョンでこれを有効にしてください。

さらに、一般的な **SSLOptions** プロパティの **CertFile** と **Password** は、PFX ファイルから証明書を読み込む際に使用されます。

Windows ストアからの証明書

証明書がすでに Windows 証明書ストアにインストールされている場合は、証明書のサムプリントを提供し、それがどこにあるかを指定してください。

サムプリントを確認するには、**PowerShell** を開いて実行してください:

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

Thumbprint 列に 必要な16進数ハッシュが表示されます。

```
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 ファイルからの証明書

PFX (.pfx または .p12) 証明書ファイルをお持ちの場合は、SSLOptionsで**CertFile** と **Password** プロパティを設定します。SChannelは起動時に証明書をインポートします。

```
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 証明書と秘密鍵をお持ちの場合は、まず OpenSSL を使用して PFX 形式に変換してください:

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

TLS バージョン

SSLOptions.Version プロパティを使用して、サーバーが受け付ける TLS バージョンを制御します:

値	説明
tls1_0	TLS 1.0 (非推奨)
tls1_1	TLS 1.1
tls1_2	TLS 1.2 (推奨)
tls1_3	TLS 1.3
tlsUndefined	TLS 1.0、1.1、1.2を受け入れます

暗号スイートリスト

デフォルトでは、SChannelはシステムの暗号構成を使用します。**SChannel_Options.CipherList**を コロン区切りのアルゴリズム名のリストに設定することで、許可される暗号を制限できます。例:

```
CALG_AES_256:CALG_AES_128
```

Windowsのデフォルトを使用するには、このプロパティを空のままにしてください。

レガシー資格情報

Windows Server 2019以前では、新しい **SCH_CREDENTIALS** APIがサポートされない場合があります。古い Windowsバージョンでサーバーの起動に失敗した場合は、レガシーの **SCHANNEL_CRED** 構造を使用するために **SChannel_Options.UseLegacyCredentials** を **True** に設定してください。

コンポーネントはほとんどの場合Windowsバージョンを自動的に検出しますが、必要に応じてレガシーモードを強制することもできます。

OpenSSL に対する利点

- **外部 DLL 不要**： SChannel は Windows に組み込まれているため、libeay32.dll / ssleay32.dll または libcrypto / libssl を配備する必要はありません。
- **Windows証明書ストアの統合**： オペレーティングシステムによってすでにインストールおよび管理されている証明書を使用します。
- **自動更新**： TLS の改善とセキュリティパッチはWindows Updateを通じて適用されます。

注意

- SChannel は**Windows のみ**で利用可能です。クロスプラットフォームサーバーには OpenSSL (iohOpenSSL) を使用してください。
- サーバーは証明書に関連付けられた**秘密鍵**を持っている必要があります。Windows ストアの方法を使用する場合、証明書は秘密鍵とともにインポートされている必要があります。
- Certificate Store方式を使用する本番サーバーでは、サービスを実行するユーザーアカウントに関係なく証明書が利用可能になるように、**Local Machine**ストア (scspStoreLocalMachine) の使用をお勧めします。
- サポートされているのは**PFX** (.pfx / .p12) 証明書 ファイルのみです。PEMファイルがある場合は、まず PFX形式に 変換してください。

TsgcWebSocketServer_HTTPAPI

HTTP Server APIを使用すると、アプリケーションはMicrosoft Internet Information Server (IIS) を使用せずに HTTP上で通信できます。

はじめに

HTTP Server API により、アプリケーションは Microsoft Internet Information Server (IIS) を使用せずに HTTP を通じて通信できます。アプリケーションは特定の URL に対する HTTP リクエストを受信するように登録し、WebSocket リクエストを受信して WebSocket レスポンスを送信できます。HTTP Server API には SSL サポートが含まれているため、アプリケーションは IIS なしでセキュアな HTTP 接続を通じてデータを交換できます。また、I/O 完了ポートと連携するように設計されています。

サーバーは以下のプロトコルをサポートします:

- WebSocket (Windows 8 以降が必要)
- HTTP 1.1
- HTTP/2 (Windows 2016以降または Windows 10以降が必要)。

デフォルトでは、このコンポーネントは URL 登録のためにアプリケーションが管理者モードで実行される必要があります。URL が netsh などの外部ツールを使用してすでに登録されている場合、管理者権限なしで実行できます。BindingOptions.ConfigureSSLCertificate プロパティを無効にして、管理者権限なしでアプリケーションを起動できるようにしてください。

FastMM4/FastMM5 をプロジェクトの最初のユニットとして設定してください。

このコンポーネントの設定手順に従います:

1. フォームにTsgcWebSocketServer_HTTPAPIコンポーネントをドロップします
2. リスニングアドレスとポートを定義します。

```
Server->Host = "127.0.0.1";  
Server->Port = 80;
```

3. 許可する仕様を設定します。デフォルトではすべての仕様が許可されています。

RFC6455： 標準かつ推奨の WebSocket 仕様です。

Hixie76： これはドラフトであり、Safari 4.2 などの古いブラウザへのサポートを提供したい場合にのみ Hixie76 接続を確立することが推奨されます

5. プロシージャを作成し、プロパティ Active = true を設定します

URL 予約

HTTP.SYS サーバーはURL予約を使用して、HTTP.SYS サーバーが使用するURLエンドポイントを割り当てます。

基本 URL 予約

これはサーバーを設定する最も簡単なシンプルなモードです。基本的にHTTP.SYSサーバーが処理するホストとポートを設定するだけです。

例: サーバーが IP 127.0.0.1 およびポート 80 で実行されている場合は、次のプロパティを設定してください。

```
Server->Host = "127.0.0.1";  
Server->Port = 80;
```

サーバーが複数の IP で動作しており、複数の IP にバインドしたい場合は、**NewBinding** メソッドを使用します。まず Host および Bindings プロパティをクリアし、NewBinding メソッドを使用してすべてのサーバーバインディングを定義します。

```
Server->Host = "";  
Server->Bindings->Clear;  
Server->Bindings->NewBinding("127.0.0.1", 80, "");  
Server->Bindings->NewBinding("80.50.55.11", 80, "");
```

サーバーが SSL 接続を必要とする場合、SSL 接続の処理に使用するホストとポートを定義するために以下を行います。

```
Server->Host = "127.0.0.1";  
Server->Port = 443;  
Server->SSL = true;  
Server->SSLOptions->Hash = "CERTIFICATE_HASH";
```

サーバーが複数の IP アドレスで SSL 接続を必要とする場合、まず Host と Bindings プロパティをクリアしてから新しいバインディングを登録してください。

```
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");
```

リファレンス

- プロパティ — コンポーネントで利用可能な27個のプロパティ。
- メソッド — 9つのパブリックメソッド。
- イベント — 処理可能な23個のイベント。

ガイド

- **設定**
 - [URL 予約](#)
- **接続**
 - [OnDisconnect が発火しない](#)
- **SSL**
 - [HTTPAPI サーバー SSL](#)
 - [自己署名証明書](#)
- **HTTP**
 - [カスタムヘッダー](#)
 - [テキストレスポンスを送信する](#)
 - [ファイルレスポンスを送信する](#)
 - [大きなファイルを投稿する](#)
- **HTTP/2**
 - [HTTP/2 の無効化](#)

HTTPAPI | URL 予約

HTTP.SYS URL 予約は Windows オペレーティングシステムの機能で、ユーザーがアプリケーションやサービスのために特定の URL を予約できます。HTTP.SYS を使用して URL が予約されると、オペレーティングシステムはその URL に対するすべての受信 HTTP リクエストをインターセプトし、指定されたアプリケーションまたはサービスにルーティングします。

HTTP.SYS を使用して URL を予約するには、アプリケーションまたはサービスがまず HTTP API を呼び出すことで URL を HTTP.SYS ドライバーに登録する必要があります。アプリケーションまたはサービスは URL、HTTP メソッド（例：GET、POST）、および認証要件などの追加設定を指定します。

URL が登録されると、HTTP.SYS はその URL のすべての受信 HTTP リクエストをインターセプトし、URL とメソッドに基づいて登録済みのアプリケーションまたはサービスを検索します。一致するアプリケーションまたはサービスが見つかった場合、HTTP.SYS ドライバーはリクエストを処理のためにそのアプリケーションまたはサービスに渡します。

NETSH Commands

URLを登録する

この例では、URL `http://example.com:80/` がユーザー `DOMAIN\user` に対して登録されています。これを任意の URL とユーザーに置き換えることができます。

```
netsh http add urlacl url=http://example.com:80/ user=DOMAIN\user
```

URLの削除

この例では、URL `http://example.com:80/` が削除されています。削除したい URL に置き換えてください。

```
netsh http delete urlacl url=http://example.com:80/
```

すべての URL を表示

このコマンドは、システム上のすべての登録済み URL 予約のリストを表示します。

```
netsh http show urlacl
```

TsgcWebSocketServer_HTTPAPI

HTTP.SYS サーバーは起動時に自動的に URL を登録します。これは以下のパラメータとメソッドを使用して行われます。

- **Host と Port:** Host が空でなく、Port がゼロでない場合、サーバーは URL を登録しようとします。例: URL `https://127.0.0.1:5000` は次のプロパティを使用して登録されます
 - Host = '127.0.0.1';
 - Port = 5000
 - SSL = True
- **NewBinding:** このメソッドを使用して1つまたは複数のURLを登録します。
 - URL `https://127.0.0.1:5000` を登録 --> `NewBinding('127.0.0.1', 5000, 'l', True)`
 - URL `http://+:5000/ws/` を登録 --> `NewBinding('+', 5000, 'ws/')`

URL 登録には次の場合に管理者権限が必要です：

- ポート番号が 1024 未満です。
- ホストは IP アドレスの代わりにワイルドカード "+" です。

ポート 443 をサーバーのすべての IP アドレスに登録し、エンドポイント「/ws/」のみでリッスンしたいが、管理者権限でサーバーを実行したくない場合は、次の手順を実行してください:

- netsh を使用して URL を登録する
 - netsh http add urlacl url=https://+:443/ws/ user=DOMAIN\user
- 以下のバインディングでサーバーを設定します。
 - NewBinding('+', 443, '/ws/', True);
- プロパティ ConfigureSSLCertificate を無効にします
 - TsgcWebSocketServer_HTTPAPI.BindingOptions.ConfigureSSLCertificate = false;
- SSL 証明書の設定
 - [HTTPAPI サーバー SSL](#)

HTTPAPI | FineTune

TsgcWSServer_HTTPAPI は **TsgcServerHTTPAPI_FineTune** 型の **FineTune** という名前の published プロパティを公開します。このプロパティは、Windows HTTP Server API (http.sys) がリクエストをキュー、ディスパッチ、完了する方法に影響するカーネルモードの低レベル設定をグループ化します。

QueueLength

HttpServerQueueLengthProperty カーネル設定をラップします。サーバーがデキューする前に **http.sys** がカーネルモードキューに保持する保留中のリクエストの最大数を制御します。キューがいっぱいになると、カーネルはユーザーモードに到達することなく、新しい接続に直接 **503 Service Unavailable** で応答します。

Type: ULONG。

Default: 1000 (Windowsカーネルのデフォルト)。

Range: 最大 65535 リクエスト。

改善される点: この値を大きくすると、バースト性のあるワークロードでカーネルが正当なトラフィックを拒否するのを防ぐことができます。ネットワーク障害後に再接続する IoT フリート、プロバイダーのリセット後に再接続するマーケットデータブロードキャストクライアント、あるいは共有のcronティックで一斉に接続するスケジュールジョブワーカーなどに有効です。

SkipIOCPOnSuccess

Trueに設定すると、サーバーはリクエストキューハンドルに対して、フラグ **FILE_SKIP_COMPLETION_PORT_ON_SUCCESS** と **FILE_SKIP_SET_EVENT_ON_HANDLE** を指定して **SetFileCompletionNotificationModes** を呼び出します。同期的に完了するリクエストキュー上のオーバーラップI/O操作は、I/O完了ポートに追加の完了パケットを投稿しなくなります。

Type: Boolean。

デフォルト: False (フラグが設定されておらず、現在の動作が維持されます)。

改善される点: 呼び出しが **NO_ERROR** を同期的に返す場合、ホットリクエストパスでのカーネルからユーザーモードへのホップを排除します。ワーカーは IOCP パケットを待たずに、呼び出し元スレッドで完了をインラインでディスパッチします。これは Microsoft のリファレンス HTTP サーバー高パフォーマンスサンプルで使用されているパターンです。このフラグは **OperatingMode = ompHighPerf** と組み合わせて使用することを目的としています。

API は **kernel32** から **GetProcAddress** を通じて実行時に解決されます。Windows XP 以前 (Delphi 7 ランタイムターゲット) ではエントリポイントが存在しないため、ラッパーは False を返し、サーバーはフラグを設定せずに実行されます。

OperatingMode

2つのaccept/dispatchアーキテクチャのいずれかを選択します。

Type: **TsgcHTTPAPIOperatingMode = (ompClassic, ompHighPerf)**。

デフォルト値: **ompClassic**。

- **ompClassic** - 単一のアクセプタースレッドが **HttpReceiveHttpRequest** を同期的に呼び出し、**PostQueuedCompletionStatus** を介して各リクエストを IOCP ワーカープールに手動でディスパッチします。これは従来の動作です。
- **ompHighPerf** - Microsoft のリファレンス HTTP サーバー高パフォーマンスパターンを実装します。アクセプタースレッドが削除されます。起動時にサーバーはキューバウンド IOCP に **ThreadPoolSize x HighPerfAcceptsPerWorker** の非同期受信を事前にポストします (デフォルト $32 \times 4 = 128$ の同時未処理受信)。各ワーカーはインラインで完了を処理し、別の非同期受信を再ポストして、シャットダウンまでスライディングウィンドウを維持します。

改善される点: **ompHighPerf** は、サーバーが深いシングルストリームパイプライン (大きなフレームのアップロード/ダウンロード) または多数の同時クライアント (数百から数千) を処理する場合に効果を発揮します。事前ポスト受信ウィンドウは接続ごとのアロケーションなしにバーストを吸収し、インラインディスパッチはアクセ

プターのハンドオフボトルネックを除去します。低トラフィック API や開発環境にはデフォルトの `ompClassic` を使用してください。軽負荷ではトラフィックの少ないワークロードで 128 の事前ポストコンテキストを維持するオーバーヘッドは節約より大きくなります。このモードはサーバーが有効化される前にのみ変更できます。

HighPerfAcceptsPerWorker

`OperatingMode = ompHighPerf` の場合に各 IOCP ワーカーが事前にポストする非同期受信数を制御します。`ompClassic` モードでは値は無視されます。サーバーが維持する同時未処理受信の合計数は `ThreadPoolSize x HighPerfAcceptsPerWorker` です。

Type: 整数。

デフォルト: 4。

改善される点: ワーカーごとのウィンドウを深く設定することで、サーバーはホットパスで新しいコンテキストを割り当てることなく、大量の着信リクエストのバーストを吸収できます。高並列デプロイメント (IoT フリート、市場データ配信、ファンアウトブローカー) では値を大きくしてください。トレードオフはメモリです。事前にポストされた各受信には、完了するまで予約されるリクエストバッファ (約 16 KB) が保持されます。デフォルトの 4 は MSDN HP サンプルに基づいて検証された保守的な中間値です。

使用例

以下のスニペットは、高並行性 IoT バックエンド向けに HTTP.sys サーバーを設定しています。再接続ストームを吸収するための大きなカーネルキュー、拡張された事前ポスト受信ウィンドウを持つ HighPerf ディスパッチ、およびインライン補完ディスパッチが有効になっています。

```
#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;
```

典型的な内部または低トラフィック API の場合、すべての `FineTune` プロパティをデフォルトのままにします:

```
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 サーバー SSL

サーバーは **SSL 証明書** を使用するように設定できます。本番サーバーにサーバー証明書を取得するには、Namecheap、GoDaddy、Thawte などの **有名なプロバイダー** から証明書を **購入** する必要があります。テスト目的では **自己署名証明書** を使用できます（自己署名証明書を使用する Demos/Chat の例を確認してください）。次の記事 [自己署名証明書の作成方法](#) をお読みください。

証明書を取得したら、接続の暗号化に使用する証明書を指定するようにサーバーを設定する必要があります。

証明書ハッシュ

まず、証明書のハッシュを知る必要があります。**powershell** では、証明書コンテナに **dir** コマンドを実行するだけで証明書のハッシュを簡単に見つけられます。

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

ハッシュは 16 進数の **Thumbprint** 値です。

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

Thumbprint の値を取得したら、**TsgcWebSocketServer_HTTPAPI.TLSOptions.Hash** プロパティに設定するだけです。

ハッシュを設定したら、**TsgcWebSocketServer_HTTPAPI.SSL = true** を設定するだけで、サーバーは起動準備が整います。

netsh を使用して証明書を手動で登録する場合は、次のコマンドを使用してください:

```
netsh http add sslcert ipport=<IP>:<PORT> certhash=<THUMBPRINT> appid="{<GUID>}"
```

<IP>: バインディングのローカル IP アドレスを指定します。ワイルドカードバインディングは使用しないでください。有効な IP アドレスを使用してください。

<PORT>: バインディングのポートを指定します。

<THUMBPRINT>: X.509 証明書のサムプリント。

<GUID>: 情報提供目的でアプリを識別するために開発者が生成した GUID。

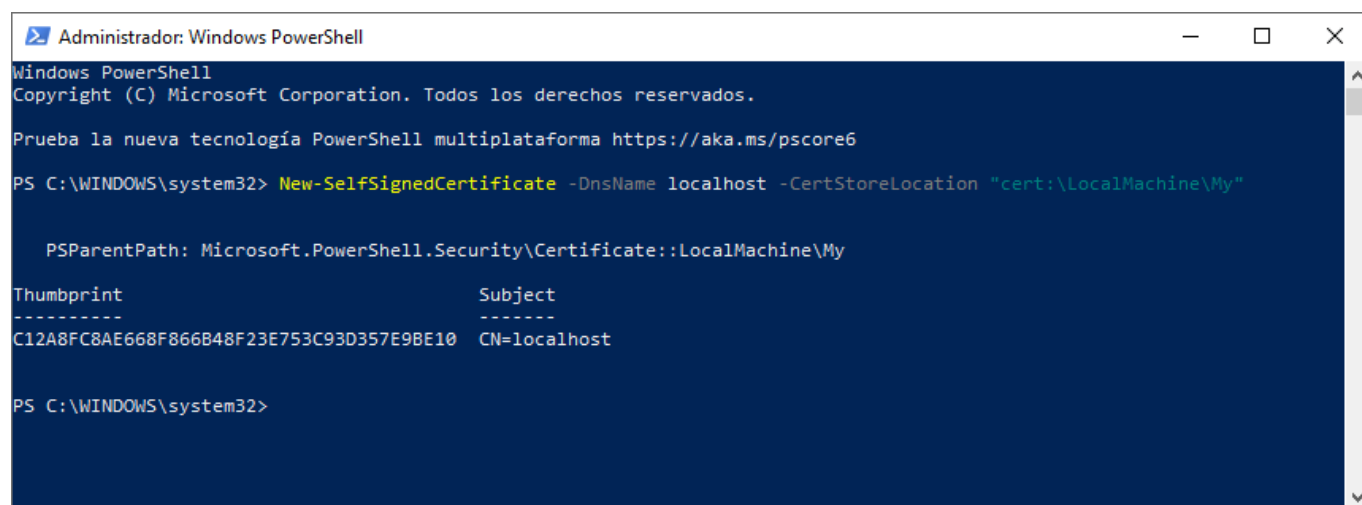
TsgcWebSocketServer_HTTPAPI | 自己署名証明書

独自のテストに証明書が必要な場合は、テストマシンで自己署名証明書を作成できます。次の手順に従います。

1. **Powershell** を**管理者**として実行します。
2. 次のコマンドを実行して証明書を作成します:

```
New-SelfSignedCertificate -DnsName localhost -CertStoreLocation "cert:\LocalMachine\My"
```

成功すると、新しい証明書の作成に関する確認が表示されます。Thumbprint をコピーして、**TsgcWebSocketServer_HTTPAPI.TLSOptions.Hash** プロパティに貼り付けてください。



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\WINDOWS\system32> New-SelfSignedCertificate -DnsName localhost -CertStoreLocation "cert:\LocalMachine\My"

PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\My

Thumbprint                               Subject
-----
C12A8FC8AE668F866B48F23E753C93D357E9BE10  CN=localhost

PS C:\WINDOWS\system32>
```

3. オプションとして、自己署名証明書を**信頼済み 証明機関**として追加できます

```
Run MMC -32 as administrator
```

- 3.1. **ファイル / スナップインの追加と削除** を選択します
- 3.2. 証明書を選択して「追加」をクリックしてください
- 3.3. コンピュータアカウントを選択し、次へをクリックします。
- 3.4. ローカルコンピューターを選択して OK を押します。これで証明書が表示されます。
- 3.5. **個人用 / 証明書**から証明書を選択し、**信頼されたルート証明機関 / 証明書**に貼り付けます。

TsgcWebSocketServer_HTTPAPI | HTTP/2 の無効化

HTTP/2 プロトコルは **Server 2016+** および **Windows 10+** OS でデフォルトで有効になっています。一部の古いブラウザや HTTP クライアントでは、プロトコルが完全にサポートされていないためエラーが発生する場合があります。HTTP/2 プロトコルを無効にすることでこれらのエラーを防ぐことができます。

HTTP/2 を無効にする方法

- Windows レジストリエディターを開く
- 以下のレジストリキーに移動します:

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\HTTP\Parameters

- 以下の DWORD 値を追加し、両方の値をゼロに設定します。
 - EnableHttp2Tls
 - EnableHttp2Cleartext
- コンピューターを再起動します。

TsgcWebSocketServer_HTTPAPI | カスタムヘッダー

HTTP.SYS サーバーのレスポンスは、レスポンスオブジェクトの **CustomHeaders** プロパティを使用してカスタマイズできます。

CustomHeaders の値をヘッダー名とヘッダー値を改行文字で区切って設定してください。

使用例: 次のヘッダーを追加する場合は、以下のサンプルコードを参照してください。

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 | テキスト 応答の送信

HTTP リクエストを処理するには **OnHttpRequest** イベントを使用します。

クラス **THttpRequest** には HTTP リクエストデータが含まれています。

- **Document**: ピアがアクセスしようとしているドキュメント。
- **Method**: HTTP メソッド ('GET', 'POST' など)。
- **Headers**: HTTP リクエストのヘッダー。
- **AcceptEncoding**: accept encoding変数。例: "gzip, deflate, br"。
- **ContentType**: 例: "text/html"
- **Content**: 存在する場合のリクエストの内容。
- **QueryParams**: クエリパラメータ。
- **Cookies**: Cookie が存在する場合の Cookie。
- **ContentLength**: コンテンツのサイズ。
- **AuthExists**, **AuthUsername**, **AuthPassword**: 認証リクエストデータ。
- **Stream**: HTTP リクエストにボディがある場合、これはボディのストリームです。

クラス **THttpResponse** は HTTP レスポンスデータを含みます。

- **ContentText**: テキストとしての応答です。
- **ContentType**: 例: 「text/html」。ContentText を UTF8 でエンコードする場合は charset='utf-8' を設定します。例: text/html; charset=utf-8
- **CustomHeaders**: 独自のヘッダーを送信する必要がある場合は、この変数を使用してください。
- **AuthRealm**: サーバーが認証を必要とする場合は、この変数を設定してください。
- **ResponseNo**: HTTPレスポンス番号。例: 200 はレスポンスが成功したことを意味します。
- **ContentStream**: レスポンスにストリームが含まれている場合は、ここに設定します（ストリームを解放しないでください。自動的に解放されます）。
- **FileName**: 応答がファイル名の場合、ここにファイル名の完全パスを設定します。
- **Date**, **Expires**, **LastModified**: レスポンスの日時変数。
- **CacheControl**: キャッシュ動作のカスタマイズを可能にします。

例: サーバーがドキュメント「/test.html」のGETリクエストを受信した場合は、OKレスポンスを送信します。そうでない場合、別のドキュメントのGETリクエストであれば404を、別のメソッドであればエラー500を送信します。

```
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 | ファイルレスポンスの送信

HTTP リクエストへの応答としてファイルを送信したい場合は、**THttpRequestResponse** オブジェクトの **FileName** プロパティを使用してください。

```
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 レスポンスは、サーバーがファイル全体ではなく、リソースの特定の部分（範囲）に対するリクエストを満たすときに使用されます。これは再開可能なダウンロード、メディアストリーミング、大容量ファイル転送に一般的に使用されます。

動作の仕組み:

クライアントが部分リソースをリクエストする場合: クライアント（ブラウザー、ダウンローダー、またはメディアプレーヤー）は、必要なバイト範囲を指定した **Range** ヘッダーを送信します。リクエスト例:

```
GET /video.mp4 HTTP/1.1
Host: example.com
Range: bytes=1000-5000
```

これは video.mp4 の1000バイト目から5000バイト目をリクエストします。

サーバーが HTTP 206 で応答: サーバーが範囲リクエストをサポートしている場合、206 Partial Content と **Content-Range** ヘッダーを含むレスポンスを返します。レスポンス例:

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 1000-5000/1000000
Content-Length: 4001
Content-Type: video/mp4
```

Content-Range ヘッダーには以下が表示されます:

提供される範囲（1000～5000）
ファイルの合計サイズ（1000000バイト）。
Content-Length ヘッダーは返される部分のサイズ（4001バイト）です。

クライアントはより多くのチャンクをリクエストできます:

クライアントは異なるパーツに対して複数のリクエストを送信できます。
これにより、再開可能なダウンロードと効率的なストリーミングが可能になります。

```

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 が発生しない

HTTPAPI サーバーを初めて使用する場合、クライアントが接続を閉じたときに OnDisconnect イベントがすぐに発火しないのは非常によくあることです。その理由は、HTTPAPI サーバーが Indy などの他のサーバーとは少し異なる動作をするためです。Indy サーバーでは、すべての接続に対してスレッドが 1 つあり、このスレッドは 接続がアクティブかどうかを x ミリ秒ごとにチェックします。HTTPAPI サーバーはすべての接続を処理するスレッドプールを使用しており、各接続がアクティブかどうかをチェックしません。

クライアントが接続を閉じたときに通知を受け取るには、次の設定を行います：

1. [TsgcWebSocketClient](#) を使用する場合は、**Options.CleanDisconnect := True** を設定してください。これにより、接続が閉じられる前に、クライアントが接続を閉じる旨のサーバーへの通知を送信しようとします。サーバーがこのメッセージを受信すると、OnDisconnect イベントが呼び出されます。
2. その他の切断については、ソケットに何かを書き込む方法しかありません。失敗した場合、接続は切断されます。HTTPAPI サーバーで **HeartBeat** を有効にし、例えば 60 秒のインターバルとタイムアウト 0 を設定します。この設定では、60 秒ごとにすべての接続に ping が送信され、切断している接続があれば **OnDisconnect** イベントが発生します。HeartBeat.Interval を低い値に設定することもできますが、あまり低くしないでください (例えば 1 秒は低すぎます)。サーバーのパフォーマンスに影響します。

TsgcWebSocketClient_WinHTTP

TsgcWebSocketClient_WinHTTP は *WebSocket* サーバーに接続できるクライアント VCL *WebSocket* コンポーネントを実装します。

はじめに

TsgcWebSocketClient_WinHTTP は *WebSocket* サーバーに接続できるクライアント VCL *WebSocket* コンポーネントを実装しています。WinHTTP API に基づいており、Windows 8 以降が必要です。このコンポーネントを設定するには以下の手順に従ってください。

1. フォームに *TsgcWebSocketClient_WinHTTP* コンポーネントをドロップします。
2. Host と Port（デフォルトは 80）を設定して、利用可能な *WebSocket* サーバーに接続します。URL プロパティを設定することで、Host、Port、パラメータが URL から自動的に更新されます。例: `wss://127.0.0.1:8080/ws/` の場合の結果:

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

3. TLS（セキュア接続）を使用するかどうかを選択できます。デフォルトでは無効です。
4. *WebSocket* クライアントのフローをカスタマイズするために次のイベントを使用できます：
5. プロシージャを作成し、Active プロパティを True に設定します。

リファレンス

- プロパティ — コンポーネントで利用可能な16のプロパティ。
- メソッド — 8個のパブリックメソッド。
- イベント — 処理可能な9個のイベント。

TsgcWebSocketFirewall

TsgcWebSocketFirewall は、WebSocketサーバーを幅広いセキュリティ脅威から保護する包括的なファイアウォールコンポーネントを実装します。

はじめに

TsgcWebSocketFirewall は、WebSocket サーバーを幅広いセキュリティ脅威から保護する包括的なファイアウォールコンポーネントを実装します。IP ブラックリスト/ホワイトリストフィルタリング、IP の自動禁止を伴うブルートフォース保護、SQL インジェクション検出、XSS（クロスサイトスクリプティング）検出、接続レート制限、メッセージフラッド保護、GeoIP 国別フィルタリング、IP ベースのボット検出、動的脅威スコアリング、パストラバーサル検出、コマンドインジェクション検出、ペイロードサイズ制限、WebSocket 固有の保護（オリジン検証、フレームサイズ制限、サブプロトコルフィルタリング）、段階的禁止エスカレーション、カスタムルールエンジン、リアルタイム統計を含む16の保護モジュールを提供します。

ファイアウォールは TsgcWebSocketServer、TsgcWebSocketHTTPServer、TsgcWebSocketServer_HTTPAPI で利用可能な **Firewall** プロパティを通じてサーバーコンポーネントと自動的に統合されます。割り当てて設定すると、手動でイベントを接続することなく、接続とメッセージをインターセプトします。

クイックスタート

1. フォームに **TsgcWebSocketFirewall** コンポーネントをドロップします。

2. 目的の保護モジュールを設定します：

ブラックリスト: Enabled を True に設定し、IP または CIDR 範囲を IP リストに追加します（例: "10.0.0.0/8"、"192.168.1.100"）。

Whitelist: Enabled を True に設定して信頼済みIPを追加します。ホワイトリストに登録されたIPはすべての他のチェックをバイパスします。

BruteForce : Enabled を True に設定してください。MaxAttempts（デフォルト 5）、TimeWindowSec（デフォルト 60）、BanDurationSec（デフォルト 300）を設定してください。

SQLInjection: Enabled を True に設定します。メッセージ内の一般的なSQLインジェクションパターンを検出します。

XSS: Enabled を True に設定します。メッセージ内のクロスサイトスクリプティングパターンを検出します。

RateLimit : Enabled を True に設定します。IP ごとの同時接続を制限します（デフォルトは 60 秒あたり 10 件）。

FloodProtection: Enabled を True に設定します。IP ごとの 1 秒あたりのメッセージ数を制限します（デフォルトは 100）。

PayloadLimit: Enabled を True に設定します。MaxSizeBytes を超えるメッセージをブロックします（デフォルト 1MB）。

PathTraversal: Enabled を True に設定します。ディレクトリトラバーサルパターンを検出します。

CommandInjection: Enabled を True に設定します。シェル コマンド インジェクション パターンを検出します。

ThreatScore: Enabledをに設定します。違反に基づいてIPに動的スコアを割り当てます。

BanEscalation : Enabled を True に設定します。常習違反者の BAN 期間をエスカレートします。

GeoIP: EnabledをTrueに設定します。国コードによって接続をブロックまたは許可します。

BotDetection: Enabled を True に設定します。接続しているIPを検証済みクローラー、データセンターホスト、疑わしいボット、ブロックリスト済み、または人間として分類します。

WebSocketProtection: Enabled を True に設定します。オリジン、フレームサイズ、サブプロトコルを検証します。

CustomRules: Enabled を True に設定します。複雑なフィルタリング用の設定可能なルール エンジンです。

3. ファイアウォールをサーバーコンポーネントに割り当てます:

```
sgcWebSocketHTTPServer1.Firewall := sgcWebSocketFirewall1;
```

4. サーバーを起動します。ファイアウォールはすべての接続とメッセージを自動的に保護します。

プロパティ

プロパティ	説明
有効	ファイアウォールを有効または無効にします。デフォルト: True。
ブラックリスト	IP ブラックリストの設定。Enabled (Boolean) と IPs (TStringList) プロパティを含みます。正確な IP と CIDR 表記 (例: 「192.168.0.0/16」) をサポートします。
Whitelist	IP ホワイトリスト設定。ホワイトリストに登録された IP はすべてのセキュリティチェックをバイパスします。Blacklist と同じ構造です。
BruteForce	ブルートフォース保護。プロパティ: Enabled、MaxAttempts (デフォルト 5)、TimeWindowSec (デフォルト 60)、BanDurationSec (デフォルト 300)。試行制限を超えた IP を自動的に BAN します。
SQLInjection	SQL インジェクション検出。プロパティ: Enabled、Action (faDeny/faAllow/faLog)、CustomPatterns (追加パターン用 TStringList)。
XSS	クロスサイトスクリプティング検出。プロパティ: Enabled、Action (faDeny/faAllow/faLog)。
RateLimit	接続レートの制限。プロパティ: Enabled、MaxConnectionsPerIP (デフォルト 10)、TimeWindowSec (デフォルト 60)。
FloodProtection	メッセージフラッド保護。プロパティ: Enabled、MaxMessagesPerSec (デフォルト 100)、Action (faDeny/faAllow/faLog)。
PayloadLimit	メッセージサイズの制限。プロパティ: Enabled、MaxSizeBytes (デフォルト 1048576)、Action (faDeny/faAllow/faLog)。
PathTraversal	パストラバーサル検出。プロパティ: Enabled、Action (faDeny/faAllow/faLog)。

CommandInjection	コマンドインジェクション検出。プロパティ: Enabled、Action (faDeny/faAllow/faLog)、CustomPatterns (追加パターン用の TStringList)。
ThreatScore	動的な脅威スコアリング。プロパティ: Enabled、AutoBanThreshold (デフォルト 80)、DecayPerHour (デフォルト 5)、スコア計算の微調整のための違反タイプごとの重み。
BanEscalation	プログレッシブな BAN エスカレーション。プロパティ: Enabled、Levels (秒単位の期間の TStringList)、ResetAfterSec (デフォルト 86400)。後続の各 BAN は次のレベルの期間を使用します。レベル値 0 は永久 BAN を意味します。
GeolIP	地理的 IP フィルタリング。プロパティ: Enabled、Mode (gmBlockList/gmAllowList)、Countries (ISO 3166-1 alpha-2 国コードの TStringList)、DatabaseFile (GeolIP CSV データベースへのパス)。
BotDetection	IP ベースのボット検出。プロパティ: Enabled、KnownBotsFile、CacheDurationSec (デフォルト 3600)、DNSTimeoutMS (デフォルト 2000)、ReverseDNS、Datacenter、DNSBL サブオブジェクト。分類のみ: IP を確認済みクローラー、データセンターホスト、疑わしいボット、ブロックリスト、または人間として分類しますが、ブロックはしません。
WebSocketProtection	WebSocket 固有の保護。プロパティ: Enabled、AllowedOrigins (TStringList)、MaxFrameSize (Integer)、AllowedSubprotocols (TStringList)。
CustomRules	カスタムルールエンジン。プロパティ: Enabled、Rules (TsgcFirewallRuleItem の TCollection)。各ルールは複雑なフィルタリングロジックの条件とアクションを定義します。

イベント

OnFiltered: 接続またはメッセージがブロックされたときに発生します。Allow パラメーター (var Boolean) でファイアウォールの決定を上書きできます。

OnViolation: セキュリティ違反が検出されたときに発生します。ログ記録のために IP アドレス、違反タイプ、詳細を提供します。

OnResolveCountry: GeolIPがIPを国コードに解決する必要があるときに発生します。カスタムの国解決を提供するにはハンドラを割り当てるか、自動検索のためにGeolIPデータベースファイルを読み込んでください。

OnBotDetected: ボット検出が IP を分類するときに発生します。ログや分析のために IP アドレス、TsgcBotClassification の判定、一致したボット名を提供します。

OnResolveBot: ボット検出がIPを分類する必要があるときに発生します。組み込みのリゾルバの代わりにカスタムの判定を提供するには、ハンドラを割り当ててください。

OnThreatScoreChanged: IP の脅威スコアが変化したときに発生します。IP アドレスと古いスコアおよび新しいスコアを提供します。

パブリックメソッド

Method	説明
IsConnectionAllowed(IP)	指定された IP からの接続が許可されているかどうかを確認します。サーバーと統合されると自動的に呼び出されます。
IsMessageAllowed(IP, Message)	指定された IP からのメッセージがすべてのメッセージフィルターを通過するかどうかを確認します。サーバーと統合されている場合は自動的に呼び出されます。
RegisterConnection(IP)	レート制限の追跡のために新しい接続を登録します。自動的に呼び出されます。
UnregisterConnection(IP)	追跡から接続を削除します。切断時に自動的に呼び出されます。
RegisterFailedAttempt(IP)	ブルートフォース検出のために失敗した認証試行を記録します。認証が失敗した場合に OnAuthentication イベントハンドラーからこれを呼び出します。
BanIP(IP, DurationSec)	IPアドレスを手動でブロックします。DurationSec = 0 は永続的なブロックを意味します。
UnbanIP(IP)	指定したIPアドレスのBANを解除します。
IsBanned(IP)	指定された IP が現在禁止されているかどうかを返します。
ClearBans	すべてのアクティブな禁止を削除します。
ClearTracking	すべての内部トラッキングデータ（接続数、試行ログ、メッセージ数）をリセットします。
SaveBansToFile(FileName)	再起動後の永続性のために、すべてのアクティブなBANをファイルに保存します。
LoadBansFromFile(FileName)	以前に保存されたファイルから BAN を復元します。
LoadGeoIPDatabase(FileName)	GeoIP CSV データベースを読み込みます。期待されるフォーマット: start_ip,end_ip,country_code。
LookupCountry(IP)	IP アドレスの 2 文字の ISO 国コードを返します。
GetBotClassification(IP)	IPアドレスのTsgcBotClassificationを返します（キャッシュ優先。キャッシュミスはbcUnknownを返し、非同期ルックアップをトリガーする可能性があります）。
GetThreatScore(IP)	IP アドレスの現在の脅威スコア（0～100）を返します。
ResetThreatScore(IP)	IP の脅威スコアを 0 にリセットします。
GetTopThreats(Count)	脅威スコアが最も高い IP のリストを返します。
IsOriginAllowed(Origin)	WebSocket Origin ヘッダーが WebSocketProtection 設定で許可されているかどうかを確認します。
IsFrameSizeAllowed(Size)	メッセージサイズが WebSocketProtection.MaxFrameSize で定義された制限内にあるかどうかを確認します。

IsSubprotocolAllowed(Subprotocol)	WebSocket サブプロトコルが WebSocketProtection 設定によって許可されているかどうかを確認します。
統計	ブロックされた接続、フィルタリングされたメッセージ、発行された禁止、脅威スコア、モジュール別ヒット数のリアルタイム カウンターを持つ、パブリック読み取り専用の TsgcFirewallStats オブジェクト。

自動統合

サーバーの Firewall プロパティに割り当てると、コンポーネントは自動的に以下を行います：

- **新しい接続時：** ブラックリスト、ホワイトリスト、BAN、レート制限、GeoIP 国をチェックし、カスタムルールを評価します。ブロックされた場合は OnTCPConnect イベントより前に接続を拒否します。
- **メッセージ受信時：** SQLインジェクション、XSSパターン、ペイロードサイズ、パストラバーサル、コマンドインジェクション、フラッド制限を確認し、カスタムルールを評価します。脅威スコアを更新します。違反が検出されるとクライアントを切断します。
- **切断時：** 内部トラッキングから接続の登録を解除します。

CIDRサポート

ブラックリストとホワイトリストは、IP範囲のCIDR表記をサポートします：

- **192.168.1.0/24** — 192.168.1.0 から 192.168.1.255 にマッチします。
- **10.0.0.0/8** — すべての 10.x.x.x アドレスに一致
- **172.16.0.0/16** — 172.16.x.x アドレスに一致

使用例

```
// 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 インジェクションパターンが検出されました。

組み込みの SQL インジェクション検出器は以下のパターン（大文字小文字を区別しない）をチェックします:

- 'OR', 'AND' — ブール インジェクション
- UNION SELECT — ユニオンベースのインジェクション
- '; DROP, '; DELETE, '; INSERT, '; UPDATE — ステートメントインジェクション
- -- (SQL comments)
- EXEC(, EXECUTE(— Command execution
- xp_cmdshell — 拡張ストアードプロシージャ
- CAST(, CONVERT(— 型変換の悪用
- 1=1, 1='1 — トートロジーインジェクション

追加のカスタムパターンは SQLInjection.CustomPatterns プロパティを通じて追加できます。

XSS Patterns Detected

- <script — スクリプトインジェクション
- javascript: — プロトコルベースの XSS
- onerror=, onload=, onclick=, onmouseover= — イベントハンドラーインジェクション
- eval(— JavaScript評価
- document.cookie — Cookie 窃取
- <iframe, <object, <embed — Element injection
- <svg onload — SVGベースのXSS
- expression(— CSS エクスプレッション インジェクション

パストラバーサルパターンが検出されました

組み込みのパストラバーサル検出器は以下のパターンをチェックします:

- ../ および ..\ — 相対ディレクトリトラバーサル
- %2e%2e%2f, %2e%2e/, ..%2f — URLエンコードされたバリエーション
- %00 — nulバイトインジェクション

- /etc/passwd — Unix 機密ファイルへのアクセス
- c:\windows — Windowsシステムディレクトリへのアクセス

コマンドインジェクションパターンが検出されました

組み込みのコマンドインジェクション検出器は以下のパターンをチェックします。

- ; | && || — シェル演算子（コマンドチェーニング）
- `command` — バックティック実行
- \$(command) — サブシェルの実行
- rm -rf — 破壊的なファイル操作
- wget, curl — リモートファイルのダウンロード
- /bin/sh, /bin/bash — Unix シェルの呼び出し
- cmd.exe — Windowsコマンドインタプリタ
- powershell — PowerShell 呼び出し

追加のカスタムパターンは CommandInjection.CustomPatterns プロパティを通じて追加できます。

スレッドセーフ

ファイアウォールコンポーネントは完全にスレッドセーフです。すべてのパブリックメソッドは内部クリティカルセクションを使用して、追跡データへの同時アクセスを保護します。20 の並行スレッドで 100,000 回の操作を実行するストレステストを行い、エラーゼロ、メモリリークゼロでした。

このコンポーネントは複数のサーバーインスタンス間で安全に共有でき、外部同期なしに任意のスレッド（サーバーイベントハンドラー、タイマースレッドなど）からアクセスできます。

リファレンス

- プロパティ — コンポーネントで利用可能な16個のプロパティ。
- メソッド — 21個のpublicメソッド。
- イベント — 処理可能なイベントが 6 つあります。

ファイアウォール: ブラックリストとホワイトリスト

TsgcWebSocketFirewall は、特定の IP をブロックするブラックリストと、信頼できる IP への無条件アクセスを付与するホワイトリストという 2 つの補完的なメカニズムによる IP ベースのアクセス制御を提供します。

ブラックリスト

ブラックリストは、指定された IP アドレスまたは IP 範囲からの接続を防ぎます。有効にすると、ブラックリストに登録された IP からの受信接続は、サーバーの接続イベントに到達する前に拒否されます。

プロパティ	説明
Blacklist.Enabled	ブラックリストチェックを有効または無効にします。デフォルト: False。
Blacklist.IPs	ブロックされた IP アドレスまたは CIDR 範囲を含む TStringList。

デザイン時に IP を追加する

オブジェクトインスペクターの IPs プロパティをクリックして文字列リストエディターを開きます。1 行ごとに 1 つの IP または CIDR 範囲を追加します：

```
192.168.1.100
10.0.0.0/8
172.16.0.0/12
```

実行時に IP を追加する

```
sgcWebSocketFirewall1->Blacklist->Enabled = true;
sgcWebSocketFirewall1->Blacklist->IPs->Add("192.168.1.100");
sgcWebSocketFirewall1->Blacklist->IPs->Add("10.0.0.0/8");
```

Whitelist

ホワイトリストは指定された IP アドレスに無条件のアクセスを付与します。ホワイトリストに登録された IP はブラックリスト、ブルートフォースブロック、レート制限、メッセージフィルタリングを含む他のすべてのファイアウォールチェックをバイパスします。

プロパティ	説明
Whitelist.Enabled	ホワイトリストチェックを有効または無効にします。デフォルト: False。
Whitelist.IPs	信頼される IP アドレスまたは CIDR 範囲を含む TStringList。

使用例

```
// 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");
```

優先順位

ブラックリストとホワイトリストの両方が有効な場合、ファイアウォールは次の順序で評価します:

1. IP がホワイトリストに登録されている場合、接続は**即座に許可**されます。それ以上のチェックは行われません。
2. IP がブラックリストに載っている場合、接続は**拒否**されます。
3. IP がどちらのリストにも含まれていない場合、接続は他のチェック（ブルートフォース、レート制限など）に進みます。

これは、ホワイトリストエントリが同じ IP のブラックリストエントリよりも常に優先されることを意味します。

CIDR 表記

ブラックリストとホワイトリストはどちらも、IP アドレス範囲を指定するための CIDR（クラスレスドメイン間ルーティング）表記をサポートしています。

CIDR	Range	アドレス
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

同じリストに完全な IP と CIDR 範囲を混在させることができます:

```
sgcWebSocketFirewall1->Blacklist->IPs->Add("203.0.113.50"); // single IP
sgcWebSocketFirewall1->Blacklist->IPs->Add("198.51.100.0/24"); // entire subnet
```

ブラックリストとホワイトリストの組み合わせ

一般的なパターンは、広い範囲をブロックしながら、その範囲内の特定の IP を許可することです:

```
// 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");
```

この例では、10.x.x.x 範囲のすべての IP がブロックされますが、10.1.1.50 はホワイトリストに登録されてすべてのチェックをバイパスします。

ファイアウォール：ブルートフォース保護

ブルートフォース保護モジュールは、同一 IP アドレスからの認証失敗の繰り返しを検出し、問題のある IP を設定可能な期間自動的にブロックします。

仕組み

ファイアウォールはスライディングタイムウィンドウ内で IP アドレスごとの失敗した試行を追跡します。失敗数が設定されたしきい値を超えると、IP は自動的にバンされます。バンされた IP は接続レベルで拒否されます（サーバーイベントが発火する前に）。

1. クライアントが接続して認証を試みます。
2. 認証に失敗した場合、コードは **RegisterFailedAttempt(IP)** を呼び出してファイアウォールに通知します。
3. ファイアウォールは現在の時間ウィンドウ内でその IP の失敗カウンターをインクリメントします。
4. カウンターが **MaxAttempts** に達すると、IP は **BanDurationSec** 秒間禁止されます。
5. BAN された IP からの新しい接続はすぐに拒否されます。
6. BAN 期間が終了すると、IP は自動的に BAN が解除され、失敗カウンターがリセットされます。

プロパティ

プロパティ	デフォルト	説明
BruteForce.Enabled	False	ブルートフォース保護を有効または無効にします。
BruteForce.MaxAttempts	5	IPが禁止される前に、時間枠内で許可される失敗試行の最大回数。
BruteForce.TimeWindowSec	60	スライディング時間枠の継続時間（秒単位）。これより古い失敗試行は破棄されます。
BruteForce.BanDurationSec	300	自動バンの期間（秒単位）。永続的なバン（手動で削除またはサーバーの再起動まで）の場合は 0 に設定します。

RegisterFailedAttempt

ファイアウォールコンポーネントは、失敗したログインがどの接続を表しているかを認識しません。ログイン試行が失敗するたびに、認証イベントハンドラーから **RegisterFailedAttempt** を呼び出す必要があります：

```
void __fastcall TForm1::ServerAuthentication(TsgcWSConnection *Connection,
String aUser, String aPassword, bool &Authenticated)
{
    Authenticated = CheckCredentials(aUser, aPassword);
    if (!Authenticated)
        sgcWebSocketFirewall1->RegisterFailedAttempt(Connection->IP);
}
```

手動 BAN 管理

自動バンに加えて、手動でバンを管理することもできます：

Method	説明

BanIP(IP, DurationSec)	IP を手動でバンします。永続的なバンには DurationSec = 0 を使用します。
UnbanIP(IP)	指定された IP の BAN を削除します。
IsBanned(IP)	IPが現在禁止されている場合はTrueを返します。
ClearBans	すべてのアクティブな禁止 (自動および手動の両方) を削除します。

例: 手動禁止

```
// 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");
```

設定例

```
// 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;
```

注意

- ホワइटリストに登録されたIPは、RegisterFailedAttemptが呼び出されても禁止されることはありません。
- デフォルトでは禁止はメモリに保存されます。サーバー再起動後も禁止を維持するには **SaveBansToFile** と **LoadBansFromFile** を使用してください。
- タイムウィンドウはスライディング方式であり、固定ではありません。各失敗試行は個別に追跡され、TimeWindowSec秒後に期限切れになります。

ファイアウォール：SQL インジェクションと XSS 検出

TsgcWebSocketFirewall は WebSocket メッセージ内の SQL インジェクションおよびクロスサイトスクリプティング（XSS）攻撃のパターンベースの組み込み検出を含んでいます。これらのモジュールは受信メッセージの内容をスキャンし、脅威パターンが検出されるとアクションを実行します。

SQL インジェクション検出

SQL インジェクションモジュールは、すべての受信 WebSocket メッセージで一般的な SQL インジェクションパターンをスキャンします。検出は大文字小文字を区別しません。

プロパティ

プロパティ	説明
SQLInjection.Enabled	SQLインジェクション検出を有効または無効にします。デフォルト：False。
SQLInjection.Action	パターンが検出されたときに取るアクション: faDeny （メッセージをブロックして切断、デフォルト）、 faLog （OnViolationイベントを発生させるがメッセージを許可）、 faAllow （アクションなし、検出は無効）。
SQLInjection.CustomPatterns	追加でチェックするための正規表現パターンの TStringList。これらは組み込みパターンに加えて評価されます。

組み込みパターン

デフォルトで以下のパターンが確認されます（大文字小文字を区別しない）：

パターン	攻撃タイプ
' OR ', ' AND '	ブールベースのインジェクション
UNION SELECT	ユニオンベースのインジェクション
'; DROP, '; DELETE, '; INSERT, '; UPDATE	ステートメントインジェクション / データ操作
-- (ダブルダッシュ)	SQL コメントインジェクション
EXEC(, EXECUTE(	コマンド実行
xp_cmdshell	拡張ストアプロシージャ悪用
CAST(, CONVERT(	型変換の乱用
1=1, 1'='1	トートロジーインジェクション

カスタムパターン

CustomPatterns プロパティを使用してアプリケーション固有のパターンを追加できます。

```
sgcWebSocketFirewall1->SQLInjection->Enabled = true;
sgcWebSocketFirewall1->SQLInjection->CustomPatterns->Add("WAITFOR DELAY");
sgcWebSocketFirewall1->SQLInjection->CustomPatterns->Add("BENCHMARK\\(");
sgcWebSocketFirewall1->SQLInjection->CustomPatterns->Add("SLEEP\\(");
```

XSS 検出

XSS モジュールは WebSocket メッセージ内のクロスサイトスクリプティングパターンを検出し、他の接続クライアントによってレンダリングされる可能性のある悪意のあるスクリプトのインジェクションを防止します。

プロパティ

プロパティ	説明
XSS.Enabled	XSS 検出を有効または無効にします。デフォルト: False。
XSS.Action	パターンが検出されたときのアクション: faDeny (ブロックして切断、デフォルト)、 faLog (OnViolation を発生させて許可)、 faAllow (アクションなし)。

組み込みパターン

以下の XSS パターンが検出されます（大文字と小文字を区別しない）：

パターン	攻撃タイプ
<script	スクリプトタグインジェクション
javascript:	プロトコルベースの XSS
onerror=, onload=, onclick=, onmouseover=	イベントハンドラーインジェクション
eval(	JavaScript code evaluation
document.cookie	Cookie の盗難
<iframe, <object, <embed	要素の注入
<svg onload	SVGベースのXSS
expression(	CSS式インジェクション

Action プロパティ

SQLInjectionとXSSモジュールの両方がTsgcFirewallAction型のActionプロパティをサポートしています。

Value	動作
faDeny	メッセージがブロックされ、クライアントが切断されます。OnViolation イベントが発生します。これがデフォルトです。

faLog	メッセージは通過が許可されますが、ログ記録のために OnViolation イベントが発生します。
faAllow	アクションは実行されません。Enabled フラグをTrueのまま保ちながら、実質的に検出を無効にします。

OnViolation イベント

SQL インジェクションまたは XSS パターンが検出された場合（Action が faDeny または faLog に設定されている場合）、OnViolation イベントが違反の詳細とともに発生します。

```
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;
    }
}
```

設定例

```
// 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;
```

注意

- パターン検出はすべての受信テキストメッセージに適用されます。バイナリメッセージはスキャンされません。
- ホワइटリストに登録されたIPは、メッセージフィルタリングを完全にバイパスします。
- 組み込みパターンは最も一般的な攻撃ベクターをカバーします。アプリケーション固有の脅威については、SQLInjection.CustomPatterns を通じてカスタムパターンを追加してください。
- 初期デプロイ中はfaLogモードを使用して、クライアントに影響を与えずに検出を監視します。その後、アプリケーションのパターンを検証したら、faDenyに切り替えます。

ファイアウォール: レート制限とフラッド防御

TsgcWebSocketFirewall はトラフィック制御のための2つの補完的なメカニズムを提供します: 接続レート制限 (IP が開くことができる接続数の制御) とメッセージフラッド保護 (接続クライアントが1秒間に送信できるメッセージ数の制御)。

接続レートリミット

レートリミットは、単一の IP アドレスが時間ウィンドウ内に確立できる接続数を制限します。これにより、単一のクライアントが利用可能なサーバー接続をすべて消費するのを防ぎます。

プロパティ

プロパティ	デフォルト	説明
RateLimit.Enabled	False	接続レート制限を有効または無効にします。
RateLimit.MaxConnectionsPerIP	10	単一の IP アドレスから許可される最大同時接続数。
RateLimit.TimeWindowSec	60	接続を追跡する時間ウィンドウ (秒単位)。これより古い接続はカウントから削除されます。

仕組み

1. 新しい接続が到着すると、ファイアウォールはその IP からのアクティブな接続数をカウントします。
2. カウントがMaxConnectionsPerIP以上の場合、新しい接続は拒否されます。
3. 接続が切断されると、追跡カウントから削除されます。
4. TimeWindowSec より古い接続はトラッキングから自動的にクリアされます。

使用例

```
// Allow maximum 5 connections per IP
sgcWebSocketFirewall1->RateLimit->Enabled = true;
sgcWebSocketFirewall1->RateLimit->MaxConnectionsPerIP = 5;
sgcWebSocketFirewall1->RateLimit->TimeWindowSec = 60;
```

メッセージフラッド保護

フラッド保護は、単一のIPが1秒あたりに送信できるメッセージ数を制限します。これにより、悪意のあるクライアントが急速なメッセージバーストでサーバーを圧倒することを防ぎます。

プロパティ

プロパティ	デフォルト	説明
FloodProtection.Enabled	False	メッセージのフラッド保護を有効または無効にします。

FloodProtection.MaxMessagesPerSec	100	単一のIPアドレスから1秒あたりに許可されるメッセージの最大数です。
FloodProtection.Action	faDeny	制限を超えた場合のアクション： faDeny （クライアントを切断）、 faLog （OnViolation を発火して許可）、 faAllow （アクションなし）。

仕組み

1. ファイアウォールは、毎秒リセットされる IP ごとのメッセージカウンターを維持します。
2. 受信した各メッセージは、送信者の IP のカウンターをインクリメントします。
3. カウンターが MaxMessagesPerSec を超えると、設定されたアクションが実行されます。
4. faDeny アクションでは、クライアントは即座に切断されます。faLog では、OnViolation イベントが発生しますが、メッセージは正常に処理されます。

使用例

```
// Limit to 50 messages per second per IP
sgcWebSocketFirewall1->FloodProtection->Enabled = true;
sgcWebSocketFirewall1->FloodProtection->MaxMessagesPerSec = 50;
sgcWebSocketFirewall1->FloodProtection->Action = faDeny;
```

両方の保護を組み合わせる

レート制限とフラッド保護が連携して包括的なトラフィック制御を提供します。

```
// 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;
```

この設定では、各 IP は最大5つの並行接続を開くことができ、それらの接続全体で合計最大 50 メッセージ/秒を送信できます。

チューニングガイドライン

シナリオ	MaxConnectionsPerIP	MaxMessagesPerSec
チャットアプリケーション	3-5	10-30
Real-time data feed	2-3	100-500
ゲームサーバー	1-2	50-100
API ゲートウェイ	10-20	200-1000

控えめな制限から始め、モニタリングに基づいて増やしていきます。faDeny を有効にする前に、トラフィックパターンを観察するために faLog モードを最初に使用します。

注意

- ホワइटリストに登録されたIPは、レート制限とフラッド保護の両方をバイパスします。
- 接続が閉じられると、レートリミット追跡データは自動的にクリアされます。すべてのカウンターを手動でリセットするには `ClearTracking` を使用してください。
- リバースプロキシの背後では、すべての接続がプロキシIPからのものとして表示される場合があります。プロキシを設定して実際のクライアントIPを渡すようにし（例: `X-Forwarded-For` ヘッダー）、サーバーが正しく解決することを確認してください。

ファイアウォール：ボット検出

TsgcWebSocketFirewall は接続している IP アドレスをボット、クローラー、データセンターホスト、または人間の訪問者として分類できます。ボット検出は、既知ボットデータベース、順方向確認済みの逆引き DNS 検証、データセンター/ASN 範囲マッチング、DNSBL ブロックリストルックアップを組み合わせて各 IP に分類を割り当てます。これは**分類専用**モジュールです。自身では接続をブロックしません。アプリケーションが何をすべきかを決定できるよう判定を報告します。

仕組み

1. IPがサーバーに接続します。
2. ファイアウォールは結果キャッシュ内のIPを検索します。キャッシュされた分類が存在し（かつ期限切れでない）場合、それがすぐに使用されます。
3. キャッシュミスの場合、IP は既知ボットファイル（確認済みクローラー）とデータセンター ASN ファイル（クラウド/ホスティング範囲）と照合されます。
4. ReverseDNS または DNSBL 検証が有効な場合、ルックアップはバックグラウンドスレッドでスケジュールされ、アクセプトスレッドをブロックしません。ルックアップが完了するまで、IPは **bcUnknown** として分類される場合があります。
5. 分類が解決されると、それは（CacheDurationSec秒の間）キャッシュに保存され、**OnBotDetected** イベントが発生します。
6. 分類はファイアウォールの Stats（fvBot 違反カウンター）を更新し、ルールの **BotType** プロパティを通じてカスタムルールで利用できます。

プロパティ

プロパティ	説明
BotDetection.Enabled	ボット検出を有効または無効にします。デフォルト値: False。
BotDetection.KnownBotsFile	検証済みクローラーのCSVファイルへのパス。各行は CIDR,botname （例: IPレンジとそれが属するクローラー）です。
BotDetection.CacheDurationSec	解決された分類が再検索される前に結果キャッシュに保持される秒数。デフォルト：3600。
BotDetection.DNSTimeoutMS	各 DNS ルックアップ（逆引き DNS および DNSBL）のタイムアウト（ミリ秒単位）。デフォルト：2000。
BotDetection.ReverseDNS.Enabled	主張されたクローラーのフォワード確認逆引き DNS（FCrDNS）検証を有効にします。デフォルト: False。
BotDetection.Datacenter.Enabled	データセンター/クラウドホスティングの範囲検出を有効にします。デフォルト: False。
BotDetection.Datacenter.ASNFile	データセンター ASN 範囲の CSV ファイルへのパス。各行は CIDR,asname （例：クラウドプロバイダーの範囲とその ASN 名）です。
BotDetection.DNSBL.Enabled	DNSBL（DNSブロックリスト）ルックアップを有効にします。デフォルト: False。
BotDetection.DNSBL.Zones	クエリするDNSBLゾーンのTStringList（例: zen.spamhaus.org ）。

分類値

TsgcBotClassification 列挙型は、IP に割り当てられた判定を表します:

Value	説明
bcUnknown	IP はまだ分類されていません。非同期ルックアップ進行中のキャッシュミス時に返されます。
bcVerifiedCrawler	既知の正規のクローラー（たとえば検索エンジンボット）で、既知ボットファイルまたはフォワード確認リバース DNS によって確認されたもの。
bcDatacenter	このIPは、ASNファイルにリストされているクラウドまたはホスティングプロバイダーの範囲に属します。多くの場合、住宅ユーザーではなく自動化されたトラフィックです。
bcSuspectedBot	その IP はボットの特徴を示していますが、検証済みクローラーとして確認できませんでした。
bcBlocklisted	そのIPは設定されたDNSBLゾーンの1つにリストされています。
bcHuman	この IP は通常の人間の訪問者のように見えます（ボット、データセンター、ブロックリストのマッチなし）。

CSV File Formats

既知のボットファイル(**KnownBotsFile**)には、検証済みのクローラー範囲が列挙されています。各行はCIDR範囲とそれに続くボット名です:

```
66.249.64.0/19,Googlebot
157.55.0.0/16,Bingbot
17.58.0.0/16,Applebot
```

データセンター ASN ファイル (**Datacenter.ASNFile**) には、クラウド/ホスティング範囲がリストされています。各行は CIDR 範囲の後に ASN 名が続きます:

```
3.0.0.0/15,AMAZON-AES
13.64.0.0/11,MICROSOFT-AZURE
34.64.0.0/10,GOOGLE-CLOUD
```

ボット検知の有効化

モジュールを有効にし、既知のボットと ASN ファイルを読み込み、必要な検証サブモジュールをオンにします。DNS ルックアップはバックグラウンドスレッドで実行されるため、ReverseDNS または DNSBL を有効にしても accept スレッドが遅くなりません。

```
// Enable bot detection
sgcWebsocketFirewall1->BotDetection->Enabled = true;
sgcWebsocketFirewall1->BotDetection->KnownBotsFile = "C:\\data\\known_bots.csv";
sgcWebsocketFirewall1->BotDetection->CacheDurationSec = 3600;
sgcWebsocketFirewall1->BotDetection->DNSTimeoutMS = 2000;

// Forward-confirmed reverse DNS verification
sgcWebsocketFirewall1->BotDetection->ReverseDNS->Enabled = true;

// Datacenter / cloud-hosting detection
sgcWebsocketFirewall1->BotDetection->Datacenter->Enabled = true;
```

```
sgcWebSocketFirewall1->BotDetection->Datacenter->ASNFile = "C:\\data\\datacenter_asn.csv";

// DNSBL blocklist lookups
sgcWebSocketFirewall1->BotDetection->DNSBL->Enabled = true;
sgcWebSocketFirewall1->BotDetection->DNSBL->Zones->Add("zen.spamhaus.org");
```

分類のみの動作と OnBotDetected

ボット検出は、接続を単独で拒否することはありません。IP が分類されると、**OnBotDetected** イベントを発生させ、ファイアウォールの統計情報（**fvBot** カウンター）を更新し、結果をキャッシュに保存します。このイベントを使用してトラフィックのログ記録、分析の構築、または独自のロジックへの入力として活用してください。

```
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;
    }
}
```

カスタムルールによる分類への対応

ボット検出は分類のみを行うため、ブロックまたは禁止はカスタムルールエンジンを通じて行われます。各 **TsgcFirewallRuleItem** は **BotType** プロパティを公開します。設定されると、ルールは分類がその値に等しい接続に一致し、拒否、禁止、またはログ記録できます。

```
sgcWebSocketFirewall1->CustomRules->Enabled = true;

// Deny connections coming from blocklisted IPs
TsgcFirewallRuleItem *vRule =
    (TsgcFirewallRuleItem *)sgcWebSocketFirewall1->CustomRules->Rules->Add();
vRule->BotType = bcBlocklisted;
vRule->ActionType = raDeny;
```

OnResolveBot によるカスタム判定

独自のボットインテリジェンス（サードパーティのレピュテーションサービス、内部の許可/拒否リストなど）をお持ちの場合は、**OnResolveBot** イベントを割り当ててください。割り当てられると、ハンドラーが組み込みリゾルバーの代わりに判定を提供します。結果を返すには **aClassification** および **aBotName** var パラメータを設定します。

```
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;
}
```

注意: OnResolveBot が割り当てられている場合、その IP に対して組み込みリゾルバー（既知のボットファイル、逆引き DNS、データセンター、DNSBL チェック）がバイパスされます。ハンドラーが返した判定はキャッシュされ、組み込みの結果と同様に OnBotDetected とカスタムルールに渡されます。

TsgcWSRateLimiter

TsgcWSRateLimiter は、サーバーエンドポイントを過剰なトラフィック、悪用、スクレイピングから保護する完全機能を備えたレート制限コンポーネントを実装します。

はじめに

TsgcWSRateLimiter は、サーバーエンドポイントを過剰なトラフィック、不正使用、スクレイピングから保護する完全機能のレート制限コンポーネントを実装します。3つのアルゴリズム (Token Bucket、Sliding Window、Fixed Window)、スコープ付きルール (IP ごと、API キーごと、ユーザーごと、エンドポイントごと)、長期クォータ (時間/日/月)、バースト保護、設定可能な HTTP 429 レスポンス形状、リアルタイム統計をサポートします。内部状態はファイルまたはストリームに永続化できるため、カウンターは再起動後も存続します。

レートリミッターは *TsgcWebSocketHTTPServer* および *TsgcWSServer_HTTPAPI* で使用可能な **RateLimiter** プロパティを通じてサーバーコンポーネントと自動的に統合されます。割り当てると、手動でイベントを配線することなく、すべての接続とすべてのメッセージを評価します。

クイックスタート

- フォームに **TsgcWSRateLimiter** コンポーネントをドロップします。
- 1つ以上のストラテジーとスコープを有効にします (各セクションのドキュメントを参照)。
- レートリミッターをサーバーコンポーネントに割り当てます:

```
sgcWebSocketHTTPServer1.RateLimiter := sgcWSRateLimiter1;
```

- サーバーを起動します。レートリミッターはすべての接続とメッセージを自動的に評価し、過剰なトラフィックを HTTP 429 で拒否し、統計カウンターを更新します。

トップレベルのプロパティ

機能の説明: コンポーネントのマスタースイッチと、状態永続化メソッドで使用されるオプションのストレージファイル。EnabledがFalseの場合、すべてのパブリックメソッドが「許可」にショートサーキットされ、カウンターは更新されません。

使用場面: 制御された負荷テストやメンテナンス期間中に、サーバーへの割り当てを解除せずにレートリミッターを実行時に無効化します。StorageFileを設定することでカウンターがアプリケーション再起動後も保持されます。例えばローリングトークンバケットは、新鮮なスタートで大量バーストを受け入れる代わりに、停止した時点から補充を続けます。

プロパティ	Type	デフォルト	説明
有効	Boolean	True	マスタースイッチ。False の場合、IsAllowed / Consume / IsConnectionAllowed / IsMessageAllowed は常に True を返し、カウンターは更新されません。

StorageFile	string	"	再起動後のカウンター永続化のために SaveStateToFile / LoadStateFromFile が使用するオプションのファイルパスです。

```
sgcWSRateLimiter1->Enabled = true;
sgcWSRateLimiter1->StorageFile = "ratelimit.dat";
sgcWSRateLimiter1->LoadStateFromFile(sgcWSRateLimiter1->StorageFile);
```

TokenBucket ストラテジー

動作:キーごとに最大Capacity個のトークンバケットを維持します。バケットはRefillIntervalMsミリ秒ごとにRefillRate個のトークンで補充されます。各リクエストは1トークン（またはConsumeに渡されたCost引数）を消費します。バケットが空になるとリクエストは拒否され、補充スケジュールからRetryAfterSecのヒントが計算されます。

使用例: API は平均 10 req/秒のレートを示していますが、行儀の良いクライアントには最大 50 リクエストのバーストを許可したい場合。Capacity=50、RefillRate=10（1000 ms あたり）のトークンバケットを使用すると、クライアントは 50 リクエストを即座に送信でき、その後バケットが補充されるまで 10 req/秒に制限されます。

プロパティ	Type	デフォルト	説明
有効	Boolean	False	Token Bucket アルゴリズムをグローバルデフォルト戦略として有効にします。
容量	Integer	100	バケットが保持できるトークンの最大数。最大バースト サイズを制御します。
RefillRate	Integer	10	RefillIntervalMs ごとにバケットに追加されるトークン数。
RefillIntervalMs	Integer	1000	ミリ秒単位の補充期間。RefillRate と合わせて持続的なリクエストレートを決定します。

```
// 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戦略

動作: すべてのリクエストにタイムスタンプを保存し、WindowSec 秒のローリングウィンドウ内に収まるタイムスタンプをカウントします。カウントが MaxRequests を超える場合は拒否します。ウィンドウは継続的にスライドするため、バーストが抜け通れる厳しいリセット時点がありません。

使用する場合: 「60 秒のローリングウィンドウ内で 100 リクエスト以下」を保証する厳格な SLA を公開しています。Fixed Window では、クライアントが 12:00:59 に 100 リクエストを送信し、12:01:00 にさらに 100 リクエストを送信できます（1 秒以内に 200 件）。Sliding Window はそれを防ぎ、SLA を正確に適用します。

プロパティ	Type	デフォルト	説明
有効	Boolean	False	グローバルデフォルト戦略として Sliding Window アルゴリズムを有効にします。
WindowSec	Integer	60	ローリングウィンドウのサイズ（秒単位）。
MaxRequests	Integer	1000	任意の WindowSec 幅のローリングウィンドウ内で許可されるリクエストの最大数。

```
// 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 戦略

動作: WindowSec 秒の固定クロックウィンドウ内のリクエストをカウントし、ウィンドウの境界でカウンターをゼロにリセットします。最もシンプルなアルゴリズムでメモリ効率が最も高いですが、境界越えの短期間に最大 2×MaxRequests を許可する可能性があります。

使用する場合: 「1 時間あたり 1000 リクエスト、カウンターは毎時リセット」のような請求スタイルのクォータ。顧客は明確に見えるリセット時刻を期待しており、請求単位が時間であってローリングの分ではないため、わずかな境界バースト動作は許容されます。

プロパティ	Type	デフォルト	説明
有効	Boolean	False	固定ウィンドウアルゴリズムをグローバルなデフォルト戦略として有効にします。
WindowSec	Integer	60	固定ウィンドウのサイズ (秒単位)。ウィンドウが経過するとカウンターは 0 にリセットされます。
MaxRequests	Integer	1000	単一の固定ウィンドウ内で許可される最大リクエスト数。

```
// Billing-style: 1000 requests per hour, resets on the hour
sgcWSRateLimiter1->FixedWindow->Enabled = true;
sgcWSRateLimiter1->FixedWindow->WindowSec = 3600;
sgcWSRateLimiter1->FixedWindow->MaxRequests = 1000;
```


PerIP

機能: ソース IP アドレスにスコープされたレートリミットを定義します。リクエストキーが PerAPIKey / PerUser / PerEndpoint に一致しない場合、PerIP がアクティブなルールになり、グローバルな Token/Sliding/Fixed のデフォルトを上書きします。各 IP は独立したカウンターを持ちます。

いつ使用するか: 多くのクライアントを持つパブリック API。PerIP なしでは、ノイジーな単一のクライアントがグローバルバジェットを消費して他全員を枯渇させる可能性があります。各 IP に独自のバケット (例: 1分間に 100 リクエスト) を与えることで、1つのクライアントが他のクライアントに影響を与えることができなくなります。

プロパティ	Type	デフォルト	説明
有効	Boolean	False	IP ごとのレートリミットを有効にします。
MaxRequests	Integer	100	ウィンドウあたりの最大リクエスト数（またはストラテジーが rlsTokenBucket の場合のバケット容量）。
WindowSec	Integer	60	ウィンドウサイズ（秒単位）。トークンバケットでは、補充レートの導出にも使用されます（MaxRequests / WindowSec トークン/秒）。
ストラテジー	TsgcRateLimitStrategy	rlsTokenBucket	このスコープに使用されるアルゴリズム: rlsTokenBucket、rlsSliding、または 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

動作内容: PerIP と同じセマンティクスですが、API キー識別子にスコープされます。コンポーネントは apikey: プレフィックスで始まるキーを API キースコープとして認識します。すべてのカスタマーがソース IP に関わらず独立したレートバジェットを持つように TsgcWSAPIKeyManager と統合するよう設計されています。

使用すべき場面: ティア別の価格設定を持つSaaS。すべてのカスタマーが独自のAPIキーと、どこから接続するか依存しない独自のクォータを持ちます。デフォルトのティアとしてPerAPIKeyを使用し、パスプレフィックスをキーとするPerEndpointルールを通じて、またはOnThrottledイベントハンドラ内で決定をオーバーライドすることで、カスタマーティア(Free / Pro / Enterprise)を区別します。

プロパティ	Type	デフォルト	説明
有効	Boolean	False	APIキーごとのレート制限を有効にします。

MaxRequests	Integer	100	ウィンドウごとの最大リクエスト数（またはトークンバケットのバケット容量）。
WindowSec	Integer	60	ウィンドウサイズ（秒単位）。
ストラテジー	TsgcRateLimitStrategy	rlsTokenBucket	このスコープに使用されるアルゴリズム。

```
// 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

機能の説明: PerIP / PerAPIKey と同じ構造ですが、アプリケーション定義のユーザー識別子をキーとして使用します。コンポーネントは user: プレフィックスで始まるキーをユーザースコープとして認識します。1 人のユーザーが複数のデバイスや IP から接続する場合に、そのユーザーの共有予算を単一に管理したいときに便利です。

使用する場面: 認証済みの Web/モバイルアプリ。ユーザー alice@example.com は電話、ラップトップ、職場のデスクトップから呼び出すかどうかに関わらず1日1000リクエストが許可されます。IPで集計すると二重カウントになります。ユーザーで集計することで、実際に課金したい予算が得られます。

プロパティ	Type	デフォルト	説明
有効	Boolean	False	ユーザーごとのレートリミットを有効にします。
MaxRequests	Integer	100	ウィンドウごとの最大リクエスト数（またはトークンバケットのバケット容量）。
WindowSec	Integer	60	ウィンドウサイズ（秒単位）。
ストラテジー	TsgcRateLimitStrategy	rlsTokenBucket	このスコープに使用されるアルゴリズム。

```
// 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

機能: パターンベースのルールのコレクション。各アイテムには独自の *Pattern*（ワイルドカード * と ? をサポート）、*Strategy*、*MaxRequests*、*WindowSec* があります。パターンがキーと一致する最初の有効なアイテムが優先されます（それ以降のアイテムは評価されません）。PerAPIKey、PerUser、PerIP より優先されます。

使用するタイミング: 高コストなエンドポイント `/api/v1/expense-report` はクライアントあたり 10 req/min に制限する必要があり、安価なステータスエンドポイント `/api/v1/status` は 1000 req/min を許可します。エンドポイントごとに異なる予算があるため、パスパターンごとに PerEndpoint アイテムを 1 つ設定してください。

プロパティ	Type	デフォルト	説明
有効	Boolean	False	PerEndpoint ルールセットを有効にします。False の場合、アイテムは評価されません。
ルール	TCollection	空	TsgcRateLimitRuleItemのコレクション。順序が重要です。最初に一致したものが優先されます。

各 **TsgcRateLimitRuleItem** は以下を公開します。

プロパティ	Type	デフォルト	説明
名前	string	"	オブジェクトインスペクターに表示されるフレンドリー名。
有効	Boolean	True	この項目を削除せずに無効化します。
パターン	string	"	キーに対して照合されるワイルドカードパターン（大文字小文字を区別しない）。* と ? をサポートします。
MaxRequests	Integer	100	ウィンドウあたりの最大リクエスト数 / バケット容量。
WindowSec	Integer	60	ウィンドウサイズ（秒単位）。
ストラテジー	TsgcRateLimitStrategy	rlsTokenBucket	一致したリクエストに使用するアルゴリズム。

```
sgcWSRateLimiter1->PerEndpoint->Enabled = true;

TsgcRateLimitRuleItem *ruleReport = (TsgcRateLimitRuleItem*)
    sgcWSRateLimiter1->PerEndpoint->Rules->Add();
ruleReport->Name = "expensive-report";
ruleReport->Pattern = "*/api/v1/expense-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;
```

クォータ

機能: メイン戦略のローリング/補充ウィンドウを超えて持続する長期的な上限です。各アイテムは、キー（またはシステム全体）がカレンダー期間（*qpHour*、*qpDay*、または *qpMonth*）内に行えるリクエスト数を制限します。*Limit* に達すると、リクエストは拒否され *OnQuotaExceeded* イベントが発生します。

使用する場面: 月あたり10,000リクエストのハードキャップに制限されたFree-tierプラン。ローリングトークンバケットは依然として短期的なトラフィックを平滑化しますが、月間クォータが使い果たされると、翌暦月までさらなる呼び出しはすべて失敗します。

プロパティ	Type	デフォルト	説明
有効	Boolean	False	クォータルールセットを有効にします。Falseの場合、アイテムは評価されません。
クォータ	TCollection	空	TsgcRateLimitQuotaltemのコレクションです。有効なすべての項目が順番に評価されます。

各 **TsgcRateLimitQuotaltem** は以下を公開します:

プロパティ	Type	デフォルト	説明
名前	string	"	OnQuotaExceededおよびResult.Reason ("quota:Name") で報告される表示名。
有効	Boolean	True	このクォータを削除せずに無効にします。
スコープ	TsgcRateLimitQuotaScope	qscPerKey	qscPerKey はキーごとに独立してカウントします。qscGlobal はすべてのキーを合わせてカウントします。
期間	TsgcRateLimitQuotaPeriod	qpHour	qpHour、qpDay、または qpMonth。カウンターは各期間の開始時に自動的にリセットされます。
制限	Int64	10000	期間内に許可される最大リクエスト数。

```
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

機能: メイン戦略と並行して実行される短期スパイク検出器です。同一キーから *BurstWindowMs* ミリ秒以内に *BurstThreshold* を超えるリクエストが到着すると、そのキーは *CooldownSec* 秒間のクールダウンに入り、その間はすべてのリクエストが拒否されます。メイン戦略が許可したとしても拒否されます。

使用する場合: 100 ミリ秒のバーストで 100 以上のリクエストでサーバーを溢れさせるが、1 分あたりの制限内に収まるスクレーパーやスキャナークライアントを検出します。バースト保護は人間のユーザーが決して起こさないこの機関銃パターンを捉えます。

プロパティ	Type	デフォルト	説明
有効	Boolean	False	バーストディテクターを有効にします。
BurstThreshold	Integer	50	クールダウンをトリガーする BurstWindowMs 内のリクエスト数。
BurstWindowMs	Integer	500	バースト観測ウィンドウ（ミリ秒単位）。
CooldownSec	Integer	30	バーストが検出された後のクールダウン期間の持続時間。クールダウン中はすべてのリクエストが拒否されます。

```
// 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;
```

レスポンス

動作内容: リクエストがスロットリングされた場合にクライアントに返される HTTP 応答の形式を設定します。ステータスコード、ボディメッセージ、および *Retry-After* と *X-RateLimit-** ヘッダーを応答に追加するかどうかを制御します。

使用するタイミング: 適切に動作するクライアントが正しくバックオフするようにしたい場合。待機時間を示す *Retry-After* ヘッダーを含む標準の 429 ステータスを返し、クライアントが拒否に反応するのではなく積極的にペースを調整できるように *X-RateLimit-Limit* / *X-RateLimit-Remaining* / *X-RateLimit-Reset* を含めてください。

プロパティ	Type	デフォルト	説明
StatusCode	Integer	429	スロットルされたリクエストに返される HTTP ステータスコード。標準の 429 またはメンテナンス形式の拒否に 503 を使用します。
メッセージ	string	"Too Many Requests"	スロットルされたリクエストに対する HTTP レスポンスのボディ。
RetryAfterHeader	Boolean	True	True の場合、計算された再試行遅延（秒）を含む Retry-After ヘッダーを追加します。
IncludeRateLimitHeaders	Boolean	True	True の場合、すべてのレスポンスに X-RateLimit-Limit / X-RateLimit-Remaining / X-RateLimit-Reset ヘッダーを追加します。

```
sgcWSRateLimiter1->Response->StatusCode = 429;
sgcWSRateLimiter1->Response->Message = "Too Many Requests - please slow down";
sgcWSRateLimiter1->Response->RetryAfterHeader = true;
sgcWSRateLimiter1->Response->IncludeRateLimitHeaders = true;
```

統計

機能: レートリミッターに関するライブ統計を公開する読み取り専用カウンターオブジェクト。すべてのカウンターは、コンポーネントの内部クリティカルセクション内でアトミックに更新されます。

使用するタイミング: 管理 UI のライブダッシュボード。TTimer から Stats をポーリングし、現在の TotalRequests / TotalThrottled / ActiveKeys をラベル、ゲージ、またはチャートに表示して、オペレーターがリアルタイムでトラフィックの圧力を確認できるようにします。

プロパティ	Type	説明
TotalRequests	Int64	最後の Reset 以降に評価されたリクエストの総数。
TotalThrottled	Int64	ストラテジーまたはバースト保護によって拒否されたリクエストの総数。
TotalQuotaExceeded	Int64	クォータに達したために拒否されたリクエストの総数。
ActiveKeys	Integer	現在追跡されているキーの数（バケット+ウィンドウカウンター+アクティブな接続）。
稼働時間	Int64	統計オブジェクトが最後にリセットされてからの秒数。

```
// 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";
}
```

イベント

OnThrottled: リクエストが戦略またはバースト保護によって拒否されたときに発生します。キー（IP、API キー、またはユーザー）と理由文字列（"token_bucket"、"sliding_window"、"fixed_window"、"burst"、"connection_rate"、または "message_rate"）を提供します。*Allow* パラメーター（var Boolean）を使用すると、決定をオーバーライドしてリクエストを通過させることができます。

OnQuotaExceeded: 長期クォータに達したときに発生します。超過したキーとクォータの名前を提供します。

OnStateChange: キーのトークンバケットサイズが変化したときに発生します。キーと古いバケットサイズおよび新しいバケットサイズを提供します。テレメトリ、デバッグ、リアルタイムビジュアライゼーションに便利です。

パブリックメソッド

Method	説明
IsAllowed(Key, Cost)	指定されたキーに対して、指定されたコスト（デフォルト 1）のリクエストが許可される場合は True を返します。
Consume(Key, Cost)	Cost トークンの消費を試み、TsgcRateLimitResult(Allowed、Remaining、RetryAfterSec、Reason)を返します。
Reset(Key)	指定されたキーのカウンターをクリアします。
ResetAll	追跡されているすべてのキーの内部カウンターをすべてクリアし、Stats オブジェクトをリセットします。
GetRemaining(Key)	指定されたキーでまだ使用可能なリクエスト数を返します。
GetRetryAfter(Key)	呼び出し元がリトライするまでに待つべき秒数を返します。
SaveStateToFile(FileName)	すべてのバケットとカウンターを指定されたファイルに永続化します。
LoadStateFromFile(FileName)	ファイルに以前保存されたバケットとカウンターを復元します。
SaveStateToStream(Stream)	内部状態を任意のストリームに保存します。
LoadStateFromStream(Stream)	任意のストリームから内部状態を復元します。
IsConnectionAllowed(IP)	サーバー統合フック。IP からの新しい接続がレート制限を通過した場合は True を返します。サーバーによって自動的に呼び出されます。

IsMessageAllowed(IP, Message)	サーバー統合フック。IP からのメッセージがレートリミットを通過した場合は True を返します。サーバーによって自動的に呼び出されます。
RegisterConnection(IP)	追跡のために新しい接続を登録します。自動的に呼び出されます。
UnregisterConnection(IP)	追跡から接続を削除します。切断時に自動的に呼び出されます。

自動統合

サーバーの RateLimiter プロパティに割り当てると、コンポーネントは自動的に以下を実行します:

- **新しい接続時:** IsConnectionAllowed を呼び出して PerIP、バースト保護、グローバル戦略を評価します。制限を超えた場合は OnConnect イベントより前に HTTP 429 で接続を拒否します。RegisterConnection を呼び出してクライアントを追跡します。
- **メッセージ受信時:** IsMessageAllowed を呼び出して、PerIP、PerEndpoint、PerAPIKey、PerUser、クォータ、バースト保護をチェックします。制限を超えた場合は切断するか、メッセージを拒否します。
- **切断時:** UnregisterConnection を呼び出してトラッキングリソースを解放します。

完全な使用例

```
// 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);
}
```

スレッドセーフ

レートリミッターは完全にスレッドセーフです。すべてのパブリックメソッドは内部クリティカルセクションとスレッドセーフなカウンターを使用して同時アクセスを保護します。コンポーネントは複数のサーバーインスタンスで共有でき、外部同期なしに任意のスレッド（サーバーイベントハンドラー、タイマースレッド、バックグラウンドワーカー）からアクセスできます。

リファレンス

- プロパティ — コンポーネントで利用可能なプロパティ13件。
- メソッド — 12 のパブリックメソッド。
- イベント — 3個の処理可能なイベント。

TsgcWSCircuitBreaker

TsgcWSCircuitBreaker は HTTP API へのクライアントサイド呼び出しにサーキットブレーカーの耐障害性パターンを実装します。

はじめに

TsgcWSCircuitBreaker は HTTP API へのクライアントサイド呼び出しのサーキットブレーカー耐障害性パターンを実装します。ローリングウィンドウでの障害を追跡し、設定可能なしきい値に達するとサーキットを開き、クールダウン期間中の呼び出しを即座に拒否することで、リモートエンドポイントが利用不能になったときのカスケード障害を防ぎます。障害数と障害率のしきい値、遅い呼び出し検出、ハーフオープントライアル呼び出し、例外分類（記録/無視）、エンドポイントごとのオーバーライド、フォールバックレスポンス、リアルタイムメトリクスをサポートします。

サーキットブレーカーは **TsgcHTTPAPI_client** (OpenAI、Anthropic、Gemini、Grok、DeepSeek、Mistral、Ollama、Stripe、Binance、Bybit、Coinbase、Kraken、その他の API ラッパー) のすべての子孫と **CircuitBreaker** プロパティを通じて自動的に統合されます。一度割り当てると、すべての Get / Post / Put / Patch / Delete 呼び出しが自動的にラップされます。一意のホスト名ごとに独立したサーキットが維持されます。

クイックスタート

1. フォームに **TsgcWSCircuitBreaker** コンポーネントをドロップします。

2. しきい値（開放するタイミング）とリカバリー（回復方法）を設定します:

```
sgcWSCircuitBreaker1.Thresholds.FailureCount := 5;  
sgcWSCircuitBreaker1.Recovery.CooldownSec := 30;
```

3. サーキットブレーカーを HTTP API クライアント（任意の *TsgcHTTPAPI_client* 派生クラス）に割り当てます:

```
sgcAI_OpenAI1.CircuitBreaker := sgcWSCircuitBreaker1;
```

4. クライアントで Get / Post / Delete / Patch / Put を呼び出します。ブレーカーは成功と失敗を自動的に記録し、サーキットが Open の間は新しい呼び出しを拒否します。

ステート

機能の説明: サーキットブレーカーは 3 状態のステートマシンです。各ホスト（キー）は独自の独立した状態を持ち、アウトバウンド呼び出しを許可するか、拒否するか、試行として送信するかを制御します。

State	動作	トランジション
csClosed	すべての呼び出しが通過します。失敗はローリングウィンドウにカウントされます。	FailureCount、FailureRatePercent、または SlowCallRatePercent に達したときに csOpen に移行します。
csOpen	すべての呼び出しは即座に拒否されます。各試行に対して OnCallRejected が発生します。	開いてから Recovery.CooldownSec 秒が経過すると csHalfOpen に移行します。

csHalfOpen	最大 Recovery.HalfOpenTrialCalls 回のトライアル呼び出しが回復を確認するために許可されます。	試行が成功すると csClosed に移行します。試行が失敗すると csOpen に戻ります。

トップレベルのプロパティ

機能の説明: コンポーネントのマスタースイッチと、呼び出しごとのキーが指定されていない場合に使用される2つのキー名。クライアント側の統合はターゲットホスト名を自動的にキーとして使用します。DefaultKeyとServerKeyは、手動のExecute/RecordSuccess/RecordFailure呼び出しおよびオプションのサーバー統合ヘルパーにのみ適用されます。

使用する場合: 制御されたロードテスト中またはメンテナンスウィンドウ中にブレーカーをランタイムで無効にします。HTTP クライアントから割り当て解除せずに行います。カスタムコード内で Execute を手動で実行する場合は、意味のある DefaultKey (例: 'openai-api') を設定してください。

プロパティ	Type	デフォルト	説明
有効	Boolean	True	マスタースイッチ。False の場合、すべての公開メソッドは「許可」に短絡し、カウンターは更新されません。
DefaultKey	string	'default'	Execute、RecordSuccess、RecordFailureのパラメータなしのオーバーロードで使用するキー。
ServerKey	string	'server'	オプションのサーバー統合フック (IsConnectionAllowed、IsMessageAllowed、RecordMessageError) で使用されるキーです。

```
sgcWSCircuitBreaker1->Enabled = true;
sgcWSCircuitBreaker1->DefaultKey = "openai-api";
sgcWSCircuitBreaker1->ServerKey = "server";
```

Thresholds

動作内容: サーキットが Closed から Open に移行する条件を定義します。FailureCount の生の障害数に達した場合、またはウィンドウ内の呼び出しの FailureRatePercent が失敗した場合 (MinCalls 後)、または呼び出しの SlowCallRatePercent が SlowCallDurationMs を超えた場合にブレーカーが開きます。

使用するタイミング: サーバーが OpenAI の API を呼び出している場合。60 秒以内に 5 回の障害が発生した場合 (または少なくとも 10 回の呼び出しが観察された後に呼び出しの 50% が失敗した場合)、サーキットを開いて OpenAI へのリクエストを停止し、リトライを重ねる代わりにリモートが回復するのを待ちます。

プロパティ	Type	デフォルト	説明

有効	Boolean	True	しきい値ベースのオープンを有効にします。False の場合、サーキットは自動的にオープンしません。
FailureCount	Integer	5	ブレーカーをトリップさせるローリングウィンドウ内の失敗の絶対数。
FailureRatePercent	Integer	50	MinCalls に達した後にブレーカーをトリガーする失敗割合（0～100）。
SlowCallDurationMs	Integer	3000	成功した呼び出しも「スロー」としてカウントされるしきい値（ミリ秒単位）。
SlowCallRatePercent	Integer	0	ブレーカーをトリップさせる遅い呼び出しの割合（0～100）。0 は遅い呼び出し検出を無効にします。
MinCalls	Integer	10	レートのしきい値を評価する前に観察される最小呼び出し数（小さなサンプルでの誤作動を防ぎます）。

```
// 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

機能: しきい値評価に使用されるローリングウィンドウを定義します。ウィンドウは BucketCount サブバケットに分割され、古いカウンターが時間の経過とともにスムーズに古くなります。しきい値は常にライブバケットの合計に対して評価されます。

使用するタイミング: 一時的なスパイクでサーキットをトリップさせず、持続的な障害でトリップさせるために、6バケットの30秒ローリングウィンドウを使用します。ウィンドウが長いほどノイズが平滑化され、短いほど障害に迅速に反応します。バケットが多いほど老化がスムーズになりますが、ホストごとのメモリ消費がわずかに増加します。

プロパティ	Type	デフォルト	説明
RollingWindowSec	Integer	60	ローリングウィンドウの合計幅（秒）。古いサンプルは破棄されます。
BucketCount	Integer	10	ウィンドウが分割されるサブバケットの数。各バケットには成功/失敗/スローのカウンターが格納されます。

```
// 30-second window split into 6 buckets (5 seconds each)
sgcWSCircuitBreaker1->TimeWindow->RollingWindowSec = 30;
```

```
sgcWSCircuitBreaker1->TimeWindow->BucketCount = 6;
```

リカバリー

機能: サーキットがOpen状態からどのように回復するかを制御します。CooldownSecが経過すると、ブレーカーはHalfOpenに遷移し、HalfOpenTrialCallsのプロープ呼び出しを通過させます。試行が成功するとブレーカーは閉じます。失敗すると再オープンし、クールダウンタイマーが再起動します。

使用する場面: 開いた後、30 秒待ってから 1 回のトライアル呼び出しを通して回復をテストし、再クローズします。長いクールダウンは苦しんでいるアップストリームを別のトラフィックの波から守ります。短いクールダウンは一時的な揺らぎのあるサービスを素早く再開します。

プロパティ	Type	デフォルト	説明
CooldownSec	Integer	30	HalfOpenに移行する前にOpen状態で待機する秒数。
HalfOpenTrialCalls	Integer	1	閉じるか再オープンするかを決定する前に、HalfOpenで許可される試行呼び出しの数。
AutoReset	Boolean	True	True の場合、状態遷移時にカウンターが自動的にクリアされます。
MaxRetries	Integer	3	リトライヘルパーが使用する情報的な上限。ブレーカー自体は自動リトライを行いません。

```
// 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;
```

フォールバック

機能: サーキットが Open 状態のときに提供される代替ペイロードを設定します。Fallback.Enabled が True で Execute / ExecuteWithResult が呼び出しを拒否すると、OnFallback イベントが CachedResponse（または CustomMessage）で初期化されたレスポンスとともに発生します。ハンドラーはこの文字列を呼び出し元に返す前に置き換えることができます。

使用するタイミング: OpenAI がダウンしているときに、ユーザーがエラーではなく機能が低下した状態（たとえば以前に生成されたサマリー）を見られるようにするために、キャッシュされた最後の正常なレスポンスを返します。静的な「メンテナンスモード」JSON ボディには CustomMessage を使用してください。

プロパティ	Type	デフォルト	説明

有効	Boolean	False	フォールバックパスを有効にします。Falseの場合、拒否された呼び出しはOnFallbackを発生させずに返るだけです。
CachedResponse	string	"	サーキットがOpenの間に返される静的ペイロード。CustomMessageより優先されます。
CustomMessage	string	"	CachedResponse が空の場合に使用されるフォールバックペイロード。
UseLastSuccess	Boolean	False	True の場合、キーに対して観察された最後の成功したレスポンスがフォールバックとして提供されます。

```
// 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;
```

分類

機能: 発生した例外を「失敗として記録」または「無視」のいずれかに分類します。記録された例外のみが失敗カウンターを進めます。MatchMode は RecordAsFailure/IgnoreExceptions の文字列が例外テキスト (ClassName: Message) と比較される方法を選択します。

使用すべき場面: 500/502/503/504を伴う EIdHttpException は失敗として扱いますが、400/401/404 の EIdHttpException は無視します(ビジネスエラーは回路を作動させるべきではありません)。RecordAsFailureを空のままにすると、無視されないすべての例外が記録されます。

プロパティ	Type	デフォルト	説明
RecordAsFailure	TStringList	空	例外テキストと照合されるパターン。一致した例外は障害としてカウントされます。空のリスト = 無視されないものはすべて記録します。
IgnoreExceptions	TStringList	空	例外テキストと照合するパターン。一致した例外は破棄されます (成功でも失敗でもありません)。
MatchMode	TsgcCircuitBreakerExceptionMatch	cemContains	比較モード: cemExact、cemContains、cemStartsWith、または 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

機能: パターンベースのオーバーライドのコレクション。各 `TsgcCircuitBreakerEndpointItem` は独自のパターン（ワイルドカード * および ? をサポート）、`OverrideThresholds` ブロック、`OverrideRecovery` ブロックを持ちます。アクティブなキー（ホスト名）が有効なアイテムに一致すると、呼び出し期間中にそのアイテムの設定がトップレベルの `Thresholds / Recovery` を置き換えます。

使用シーン: APIごとに別々のサーキット: `api.openai.com` は緩やかなしきい値（負荷時に偶発的に遅くなる場合があります）ですが、`api.stripe.com` は決済に重要なため、フェイルクローズドを速くするために早く開くべきです。ホストごとに調整されたしきい値を持つエンドポイントアイテムを1つ追加します。

プロパティ	Type	デフォルト	説明
有効	Boolean	False	エンドポイントオーバーライドコレクションを有効にします。Falseの場合、アイテムは評価されません。
アイテム	TCollection	空	<code>TsgcCircuitBreakerEndpointItem</code> のコレクション。最初に有効な一致が優先されます。

各 `TsgcCircuitBreakerEndpointItem` は以下を公開します：

プロパティ	Type	デフォルト	説明
パターン	string	"	キー（ホスト名）に対してマッチするワイルドカードパターン。* と ? をサポートします。
有効	Boolean	True	この項目を削除せずに無効化します。
OverrideThresholds	TsgcCircuitBreakerThresholds	defaults	パターンが一致したときに適用されるしきい値。それ自身の <code>Enabled</code> が <code>True</code> の場合にのみアクティブです。
OverrideRecovery	TsgcCircuitBreakerRecovery	defaults	パターンが一致したときに適用されるリカバリ設定。

```
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;
```

メトリクス

機能: ブレーカーを通じて呼び出しが流れるにつれて原子的に更新される読み取り専用カウンター。すべてのキー/ホスト名にわたって集計されます。CurrentOpenBreakers はすべての状態遷移後に更新されます。AverageLatencyMs は Execute によって観察された継続時間の実行平均です。

使用場面: ホストごとのサーキット状態、障害率、拒否数を表示するライブダッシュボード。TTimer から Metrics をポーリングしてオペレーター向けにゲージを表示するか、アラート用に Prometheus / StatsD に出力します。

プロパティ	Type	説明
TotalCalls	Int64	処理された呼び出しの総数（成功 + 記録された失敗）。
TotalFailures	Int64	すべてのキーにわたって記録された合計失敗数。
TotalSuccesses	Int64	すべてのキーにわたる成功した呼び出しの合計数。
TotalRejected	Int64	サーキットがオープンだったために拒否された呼び出しの総数。
CurrentOpenBreakers	Integer	現在 Open 状態にあるキーの数。
AverageLatencyMs	Double	Execute / ExecuteWithResultの呼び出しの周辺で測定された実行中の平均レイテンシ(ミリ秒)。

```
// 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";
}
```


イベント

OnStateChange: サーキットが状態間を遷移したときに発生します。キー、古い状態、および新しい状態 (csClosed、csOpen、またはcsHalfOpen) を提供します。ログ記録、アラート、ライブUIの更新に便利です。

OnCallRejected: サーキットが Open またはハーフオープンの特ライアル予算が枯渇したためにコールが拒否されたときに発生します。キーと理由文字列 (例: Circuit Open) を提供します。

OnFallback: フォールバックレスポンスが返される直前に発火します。*aResponse* パラメーター (var string) は Fallback.CachedResponse / CustomMessage から初期化され、ハンドラーによって置き換えることができます。

OnFailureRecorded: 失敗が記録されるたびに発生します。キーと例外テキスト (ClassName: Message) を提供します。構造化ログや上流のインシデントトラッキングとの関連付けに使用します。

OnSlowCall: 成功した呼び出しがThresholds.SlowCallDurationMsを超えたときに発火します。キーと測定された時間 (ミリ秒単位) を提供します。

パブリックメソッド

Method	説明
Execute(Key, Action)	指定されたキーのTProc を実行します。最初に IsCallAllowed を確認し、成功/失敗/低速を自動的に記録して、アクションが実行された場合はTrueを返します。
Execute(Action)	DefaultKeyを使用したオーバーロード。
ExecuteWithResult(Key, Action, out Result)	(D2009+) TFunc<TObject>を実行し、その結果をout引数経由で返します。Executeと同じアカウンティングを行います。
RecordSuccess(Key)	成功した呼び出しを記録します。試行が成功するとHalfOpen -> Closedへ進みます。
RecordSuccess	DefaultKeyを使用したオーバーロード。
RecordFailure(Key, Exception)	オプションの例外テキストとともに失敗を記録し、状態遷移の評価をトリガーします。
RecordFailure(Exception)	DefaultKeyを使用したオーバーロード。
ForceOpen(Key)	サーキットを強制的にOpen状態にします。
ForceClose(Key)	サーキットを強制的にClosed状態にします。
Reset(Key)	指定されたキーのカウンターと状態をクリアします。
ResetAll	追跡されているすべての回路をクリアし、Metricsをリセットします。
GetState(Key)	キーの現在の TsgcCircuitState を返します。

IsCallAllowed(Key)	新しい呼び出しが現在許可されているかどうかを返します。すべてのリクエストの前にHTTP APIクライアントによって自動的に呼び出されます。
GetFailureRate(Key)	ローリングウィンドウで観測された現在の障害率（0.0 ～ 100.0）を返します。
GetTotalCalls(Key)	キーのローリングウィンドウで観測された呼び出し数を返します。
GetLastStateChange(Key)	最後の状態遷移の TDateTime を返します。
GetOpenBreakers	現在 Open 状態にあるキーの TArray<string> を返します。
SaveStateToFile(FileName)	すべてのサーキットの状態とカウンタをファイルに永続化します。
LoadStateFromFile(FileName)	以前に保存したファイルからサーキット状態を復元します。
IsConnectionAllowed(IP)	サーバーサイドの自己保護フック。クライアントサイドの主要な使用に対する二次的な手段です。ServerKey サーキットが Open でない場合に True を返します。
IsMessageAllowed(IP, Message)	サーバー側の自己保護フック。ServerKey サーキットがオープンでない場合は True を返します。
RegisterConnection(IP)	将来のIPごとのメトリクス用に予約されたサーバー側フック（現在は何もしません）。
UnregisterConnection(IP)	将来のIPごとのメトリクス用に予約されたサーバー側フック（現在は何もしません）。
RecordMessageError(IP, Exception)	サーバーサイドの自己保護フック。ServerKey の障害を記録します。
RecordMessageSuccess(IP)	サーバー側の自己保護フック。ServerKey の成功を記録します。

自動統合

TsgcHTTPAPI_client の子孫に CircuitBreaker プロパティを通じて割り当てられると、クライアントは自動的に以下を行います。

- **すべてのGet / Post / Put / Patch / Delete の前:** ターゲットURLのホスト名（例: api.openai.com）からキーを導出し、IsCallAllowed(host)を呼び出します。サーキットがOpen状態の場合、呼び出しは例外 *"Circuit breaker open for <host> - call rejected"* で拒否され、OnCallRejectedが発生します。
- **HTTPレスポンスが成功した場合:** RecordSuccess(host)を呼び出してカウンターを更新し、リモートが回復したときにHalfOpenトライアルが回路を閉じられるようにします。
- **例外発生時:** RecordFailure(host, exception.Message)を呼び出し、Classification ルールを適用してカウントするか無視するかを決定し、遷移を評価してから、例外を呼び出し元に再スローします。
- **ホストごとに1つのサーキット:** 異なるホスト（例えばapi.openai.com、api.anthropic.com、api.stripe.com）への呼び出しは、完全に独立したサーキットを追跡します。

完全な使用例

```

// 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);
}

```

スレッドセーフ

サーキットブレーカーは完全にスレッドセーフです。すべてのパブリックメソッドは内部クリティカルセクションとスレッドセーフリストを使用して、キーごとの状態、ローリングウィンドウ、メトリクスカウンタへの同時ア

アクセスを保護します。単一のコンポーネントを複数の HTTP API クライアントで共有し、任意のスレッド (イベントハンドラー、タイマー、バックグラウンドワーカー) から外部同期なしにアクセスできます。

リファレンス

- プロパティ — コンポーネントで利用可能な 10 個のプロパティ。
- メソッド — 17 の公開メソッド。
- イベント — 処理可能なイベントが 5 つあります。

TsgcWSAPIKeyManager

TsgcWSAPIKeyManager は、*sgcWebSockets* サーバーが発行する API キーのライフサイクル全体の管理を提供します。

はじめに

*TsgcWSAPIKeyManager*は、*sgcWebSockets*サーバーによって発行されるAPIキーのフルライフサイクル管理を提供します。設定可能な人間が読めるプレフィックスを持つ暗号的にランダムなキーを生成し、SHA-256、SHA-512、またはBcryptで保存時にハッシュ化し、ヘッダーまたはクエリ文字列によって着信リクエストを検証し、設定可能な猶予期間でキーをローテーションし、キーを失効・更新し、スコープベースの認可を強制し、永続的な監査ログを維持し、オプションのIP許可リストをサポートします。キーはメモリ内、または再起動を生き延びる暗号化されたファイルに保存されます。

マネージャーは **APIKeyManager** プロパティを通じて *TsgcWebSocketHTTPServer* および *TsgcWSServer* HTTPAPI にアタッチします。サーバーの自動フックはピア IP しか確認できないため、実際のリクエスト認証は *OnConnect* イベントから **IsRequestAuthorized** を呼び出すことで行われます。これにより完全な HTTP ヘッダーブロックと URL/クエリ文字列へのアクセスが得られ、設定された **X-API-Key** ヘッダー（またはクエリパラメーター）を解析、検証し、必要に応じて必要なスコープと照合できます。

クイックスタート

- フォームに **TsgcWSAPIKeyManager** コンポーネントをドロップします。
- 生成（プレフィックス、長さ）、ハッシュ（アルゴリズム、ソルト）、検証（ヘッダー名、IP アローリスト、FailClosed）を設定してください。各セクションのドキュメントを参照してください。
- マネージャーをサーバーコンポーネントに割り当てます:

```
sgcWebSocketHTTPServer1.APIKeyManager := sgcWSAPIKeyManager1;
```

- 接続のヘッダー、URL、IP を使用して **IsRequestAuthorized** を呼び出すようにサーバーの *OnConnect* をワイヤリングします。False が返された場合はクライアントを切断します。
- 顧客に最初のキーを発行し、平文の値を一度だけ届けます。その後はハッシュ化された形式のみがストレージに残ります。

トップレベルのプロパティ

機能の説明: コンポーネント全体のマスタースイッチ。Enabled が False の場合、すべてのパブリックメソッドは「許可」として短絡し、サーバーの自動フックはノーオペレーションになるため、キーマネージャーは実質的にバイパスされます。

使用する場面: コンポーネントをサーバーから割り当て解除することなく、計画的なメンテナンスウィンドウやステージングデプロイ中にキーの強制を一時的に無効化します。変更が完了したら再度有効にしてください。キーは失われず、スコープも変わらず、フリップバックした瞬間から監査ログも継続して動作します。

プロパティ	Type	デフォルト	説明

有効	Boolean	True	マスタースイッチ。False の場合、ValidateKey / IsRequestAuthorized は常に True を返し、サーバーの自動フックはノーオペレーションになります。

```
sgcWSAPIKeyManager1->Enabled = true;
```

Generation

機能: *IssueKey* によって平文キー文字列が構築される方法を制御します。キーは *KeyPrefix* の後に *Charset* からランダムに選ばれた *KeyLength* 文字が続く形式です。IncludeChecksum が True の場合、短いチェックサムが付加されるため、ハッシュ検索前に不正なキーを安価に拒否できます。

使用する場面: SaaS 用のブランデッドキー — 本番環境では *sgc_live_aB3...* を、ステージングクラスターでは *sgc_test_...* を使用します。プレフィックスにより、サポートスタッフが漏洩したキーがどの環境に属するかを容易に識別でき、ログスクラバーがそれを編集できます。

プロパティ	Type	デフォルト	説明
有効	Boolean	True	False の場合、IssueKey は新しいキーを生成する代わりに例外を発生させます。
KeyPrefix	string	"	ランダムな本文の先頭に付加される、人間が読める接頭辞です (例: <i>sgc_live_</i>)。
KeyLength	Integer	32	キーの本体のランダム文字数。
文字セット	string	A-Za-z0-9	ランダムなボディを生成するために使用されるアルファベット。空の場合は URL セーフな英数字がデフォルトになります。
IncludeChecksum	Boolean	False	True の場合、明らかに不正なキーをハッシュルックアップなしで拒否できるよう、短いチェックサムサフィックスを付加します。

```
// Branded production keys: sgc_live_aB3...
sgcWSAPIKeyManager1->Generation->Enabled = true;
sgcWSAPIKeyManager1->Generation->KeyPrefix = "sgc_live_";
sgcWSAPIKeyManager1->Generation->KeyLength = 32;
sgcWSAPIKeyManager1->Generation->IncludeChecksum = true;
```

Storage

機能: ハッシュ化されたキーと監査ログがどこに保存されるかを制御します。 *kstMemory*はすべてをプロセス内に保持します(シャットダウン時にクリアされます)。 *kstFile*は *FileName*に永続化し、オプションで *EncryptionKey*で暗号化し、 *AutoSaveSeconds*秒ごとに再保存します。 *kstCustom*では、独自の *SaveToFile* / *LoadFromFile*フックを実装できます。

使用するタイミング: 高速な反復のために開発中はキーをメモリに保持します。キーがサーバーの再起動後も持続し、ディスク上に平文で公開されないように、本番環境では *EncryptAtRest := True* を指定した *kstFile* に切り替えます。

プロパティ	Type	デフォルト	説明
StorageType	TsgcAPIKeyStorageType	kstMemory	kstMemory (揮発性)、kstFile (FileNameに永続化)、または kstCustom (ユーザー提供のSaveToFile / LoadFromFile)。
FileName	string	"	StorageType が kstFile の場合に SaveToFile / LoadFromFile が使用するファイルパス。
EncryptAtRest	Boolean	False	True の場合、EncryptionKey を使用してディスク上のキーストアを暗号化します。
EncryptionKey	string	"	永続化されたキーファイルを暗号化するために使用するシークレット。EncryptAtRest が True の場合に指定する必要があります。
AutoSaveSeconds	Integer	0	0 より大きい場合、マネージャーは AutoSaveSeconds 秒ごとにキーストアをディスクに再保存します。

```
// 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");
```

ハッシュ化

機能: 平文のキーは決して永続化されません。 *IssueKey*は、生成されたキーを *Algorithm*とオプションの *Salt*および *Iterations*カウントを使用してハッシュ化します。 *ValidateKey*は着信値を再ハッシュ化し、保存されたダイジェストに対して定数時間の比較を実行します。

使用すべき場面: 平文を保存しないでください。SHA-256 はほとんどのデプロイメントで適切なデフォルトです。より高いセキュリティプロファイル (FIPS 140-2 レベル 3) には SHA-512 を使用してください。意図的なキーストレッチングが必要な場合は Bcrypt を使用してください。検証は遅くなりますが、漏洩したキーストアに対するオフラインブルートフォースが格段に困難になります。

プロパティ	Type	デフォルト	説明
アルゴリズム	TsgcAPIKeyHashAlgorithm	khaSHA256	保存時のストレージに使用されるハッシュ関数：khaSHA256、khaSHA512、またはkhaBcrypt。
Salt	string	"	すべてのハッシュに混合される静的ソルト。変更するとすべての既存のキーが無効になります。
反復回数	Integer	1	ハッシュラウンドの数。検証の遅延を犠牲にしてキーストレッチングを行うには増やします。

```
// 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;
```

スコープ

機能: 許可されるスコープ文字列のカatalog。EnabledがTrueの場合、発行されたキーにアタッチされるすべてのスコープがこのリストに存在する必要があります。不明なスコープは拒否されます。個々のCatalogエントリは、削除せずに無効化できます。ValidateKey / IsRequestAuthorizedは、リクエストごとに特定のスコープを要求できます。

使用すべき場面: きめ細かい権限。あるカスタマーのキーはread:ordersのみを持ち、書き込みエンドポイントでは拒否されます。パートナー統合はread:orders + write:shipmentsを受け取りますが、admin:*は受け取りません。各リクエストはスコープヒントを提示し、マネージャーはそれを一貫して強制します。

プロパティ	Type	デフォルト	説明
有効	Boolean	True	スコープの強制を有効にします。False の場合、スコープパラメータは無視されます。
スコープ	TCollection	空	許可されたスコープを定義する TsgcAPIKeyScopesItem エントリのコレクションです。

各 TsgcAPIKeyScopesItem は以下を公開します:

プロパティ	Type	デフォルト	説明
名前	string	"	スコープ識別子 (例: read:orders)。大文字小文字を区別します。
説明	string	"	オブジェクトインスペクターおよび管理UIに表示される人間が読める説明。

有効	Boolean	True	このスコープを削除せずに無効にします。無効になったスコープは HasScope を満たしません。

```
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";
```

ローテーション

機能の説明: キーのローテーションを透過的に行います。RotateKey は同じ所有者とスコープに対して新しいキーを発行し、古いキーに *kksRotated* マークを付けます。GracePeriodSec の間、両方のキーが有効となるため、ダウンタイムなしにキーを切り替えることができます。AutoRotateDays が 0 より大きい場合、そのしきい値を超えたキーはバックグラウンドスイープによってローテーションされます。NotifyBeforeExpirySec を設定すると OnKeyExpired が早期に発火するため、顧客にメールを送信できます。

使用するタイミング: 顧客がキーが漏洩したと報告したが、その統合はまだ稼働中の場合。キーをローテートします。顧客は新しいキーを受け取り、古いキーはインフラ全体にロールアウトできるよう24時間有効なまま保持され、その後古いキーは自動的に無効化されます。

プロパティ	Type	デフォルト	説明
有効	Boolean	False	ローテーション機能（猶予期間と自動ローテーションスイープ）を有効にします。
GracePeriodSec	Integer	86400	RotateKey 後に以前のキーが有効である秒数。デフォルトは 24 時間です。
AutoRotateDays	Integer	0	0 より大きい場合、バックグラウンドスイープはこの日数より古いキーをローテートします。
NotifyBeforeExpirySec	Integer	604800	キーが実際に期限切れになる何秒前に OnKeyExpired を発生させるか。デフォルトは 7 日間。

```
// 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;
```


有効期限

機能: 発行されたキーの期限切れを制御します。*DefaultTTLSec* は、呼び出し元が *ExpiresInSec* に 0 を渡したときに *IssueKey* に適用されます。*EnforceExpiry* は期限切れのキーを拒否するかどうかを切り替えます。*CheckIntervalSec* は、バックグラウンドスweepが期限切れエントリをスキャンして *OnKeyExpired* を発生させ *Stats* を更新する頻度を制御します。

使用する場合: パートナー統合キーは90日後に期限切れにする必要があります。パートナーは契約に再署名し、新しいキーを取得する必要があります。*DefaultTTLSec* を 90×86400 に設定すると、呼び出し元が明示的にオーバーライドしない限り、すべての *IssueKey* 呼び出しがそのポリシーを尊重します。

プロパティ	Type	デフォルト	説明
DefaultTTLSec	Integer	0	新しく発行されたキーのデフォルトの有効期間（秒単位）。0 は期限切れなしを意味します。
EnforceExpiry	Boolean	True	True の場合、 <i>ValidateKey</i> は <i>ExpiresAt</i> が過去の日付のキーを拒否します。
CheckIntervalSec	Integer	60	バックグラウンドのスweepが期限切れのキーをスキャンして <i>OnKeyExpired</i> を発生させる頻度。

```
// 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

機能: 各キーとともに、キーごとのレート制限のデフォルト(ウィンドウあたりの最大リクエスト数)を保存します。これらの値は情報提供のヒントであり、*TsgcWSRateLimiter* インスタンスが *PerAPIKey* スコープを通じて読み取り、ティアに応じたスロットリングを適用できます。APIキーマネージャー自体はトラフィックをスロットリングしません。

いつ使用するか: 階層型 SaaS。Free ティアの顧客は 100 req/時間、Pro ティアは 10,000 req/時間、Enterprise は 1,000,000 req/時間 でプロビジョニングされます。API キー マネージャー で *DefaultMaxRequests* / *DefaultWindowSec* を設定し、*TsgcWSRateLimiter.PerAPIKey* とペアにしてレートリミッターがキーごとのバジェットを自動的に読み取れるようにします。

プロパティ	Type	デフォルト	説明
有効	Boolean	False	発行されたキーごとにレートリミットのメタデータを有効にします。
DefaultMaxRequests	Integer	100	新しいキーに割り当てられるデフォルトの最大リクエスト数 (ウィンドウあたり)。

DefaultWindowSec	Integer	60	DefaultMaxRequests に関連付けられたデフォルトウィンドウサイズ (秒)。

```
// 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
```

監査

機能概要: すべてのキーライフサイクルアクション（発行、検証、失効、ローテーション、期限切れ、スコープ付与/失効、検証失敗）をインメモリリングバッファに記録し、*LogFile* が設定されている場合はファイルにも同じエントリを追記します。*IncludeIP* はリクエスターの IP を記録し、*IncludePayload* は追加の詳細をキャプチャします。*RetentionDays* は古いエントリを削除し、*MaxMemoryEntries* はリングバッファの上限を設定します。

使用する場合: コンプライアンス要件: すべてのキー操作はログに記録し、12か月間保持する必要があります。SIEMのドロップゾーンを指すLogFileとRetentionDays=365を設定して監査を有効にします。各エントリに対してOnAuditEventが発生するため、メッセージバスに送信することもできます。

プロパティ	Type	デフォルト	説明
有効	Boolean	True	監査ログを有効にします。Falseの場合、エントリは記録されず、OnAuditEventは発生しません。
LogFile	string	"	監査ログが追記されるオプションのファイルパス。
IncludeIP	Boolean	True	Trueの場合、すべてのエントリが要求者のIPを記録します。
IncludePayload	Boolean	False	Trueの場合、各監査エントリの Details フィールドに追加のコンテキストが含まれます。
RetentionDays	Integer	90	バックグラウンドスweepによって、この日数より古いエントリが削除されます。
MaxMemoryEntries	Integer	10000	メモリ内のリングバッファに保持されるエントリの最大数。

```
// 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;
```

検証

機能: クライアントがキーを提示する方法とリクエストが受け入れられる条件をマネージャーに伝えます。
HeaderName (デフォルト *X-API-Key*) と *QueryParamName* (デフォルト *api_key*) は *ExtractKeyFromHeaders* / *ExtractKeyFromQuery* が参照する 2 か所です。*RequireHTTPS* はプレーンテキストトラフィックを拒否します。
IPAllowlist はキーを特定のソース IP に制限します。*FailClosed* (デフォルト *True*) はマネージャーが確実に認可できないリクエストを拒否します。

使用場面: 社内専用 API: すべてのリクエストに HTTPS を要求し、キーをオフィスの出口 CIDR 範囲に制限することで、キーが漏洩しても社外からの悪用を防止します。

プロパティ	Type	デフォルト	説明
HeaderName	string	X-API-Key	受信キーを読み取るために使用されるHTTPヘッダー名。
QueryParamName	string	api_key	ヘッダーが存在しない場合のフォールバックとして使用されるクエリ文字列パラメータ。
RequireHTTPS	Boolean	False	True の場合、キーの有効性に関わらずすべての非 HTTPS リクエストを拒否します。
IPAllowlist	TStringList	空	空でない場合、リスト内の IP を持つリクエストのみが承認されます。
FailClosed	Boolean	True	Trueの場合、マネージャーが明示的に認可できないリクエストはすべて拒否されます。

```
// 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");
```

統計

機能の説明: キーストアに関するライブ統計を公開する読み取り専用のカウンターオブジェクトです。すべてのカウンターはコンポーネントの内部クリティカルセクション内でアトミックに更新されます。

いつ使用するか: ライブ管理ダッシュボード。TTimer から Stats をポーリングして、ActiveKeys / RevokedKeys / TotalValidations / TotalValidationFailures を表示することで、オペレーターがリアルタイムで認証の負荷を確認できます。

プロパティ	Type	説明
TotalKeys	Int64	このマネージャーインスタンスが発行したキーの総数。

ActiveKeys	Int64	現在 kksActive ステータスのキーの数。
RevokedKeys	Int64	ステータス kksRevoked のキーの現在の数。
ExpiredKeys	Int64	ステータス kksExpired のキーの現在の数。
TotalValidations	Int64	ValidateKey / IsRequestAuthorized 呼び出しの合計数。
TotalValidationFailures	Int64	キーを拒否した検証呼び出しの総数。
LastIssuedAt	TDateTime	最新のIssueKey呼び出しのタイムスタンプです。

```
// 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);
}
```

イベント

OnKeyIssued: 新しいキーが発行されたときに発生します。オーナー、生のプレーンテキストキー (最後の観察機会)、スコープ配列を提供します。このハンドラーからプレーンテキストをお客様に配信してください。

OnKeyValidated: ValidateKey が完了するたびに発火します。キー、ブール値の *aValid* フラグ、機械可読な理由文字列を提供します。

OnKeyRevoked: キーが失効したときに発火します。キーと失効理由を提供します。

OnKeyRotated: キーがローテーションされたときに発生します。古いキーと新しいキーを提供して、顧客に通知できます。

OnKeyExpired: バックグラウンドの有効期限スリープによって、キーの ExpiresAt が経過した (または NotifyBeforeExpirySec 秒前に) 発生します。

OnValidation: ValidateKey 中に発生する最終決定フック。Allow := False を設定し、Reason を指定することで、それ以外は有効なキーを拒否できます (カスタムビジネスルール)。

OnAuditEvent: すべての監査ログエントリに対して発生します。TsgcAPIKeyAuditEntry レコード (Timestamp、KeyPrefix、Action、IP、Details) を提供します。外部 SIEM への転送に最適です。

パブリックメソッド

Method	説明
IssueKey(Owner, Scopes, ExpiresInSec)	新しいキーを生成、ハッシュ化、保存します。プレーンテキストを返します (見られるのはこの 1 回のみ)。ExpiresInSec = 0 を渡すと Expiration.DefaultTTLSec が継承されます。

ValidateKey(Key, RequiredScope, IP)	生のキーを検証します。オプションで必要なスコープを適用し、リクエストの IP を記録します。成功した場合は True を返します。
RevokeKey(Key, Reason)	キーを失効済みとしてマークします。その後の ValidateKey 呼び出しは False を返します。
RotateKey(OldKey, out NewKey)	同じオーナーとスコープに対して新しいキーを発行します。古いキーは kksRotated としてマークされ、GracePeriodSec の間は引き続き有効です。
RenewKey(Key, NewExpiresInSec)	キーの有効期限を現在から指定された秒数だけ延長します。
GetKeyInfo(Key)	所有者、スコープ、ステータス、タイムスタンプ、使用回数を含む TsgcAPIKeyInfo レコードを返します。
GetKeysByOwner(Owner)	指定されたオーナーに関連付けられたハッシュキー識別子を返します。
ListScopes(Key)	キーに付与されたスコープを返します。
IsExpired(Key)	キーが有効期限を過ぎている場合は True を返します。
IsRevoked(Key)	キーが失効している場合は True を返します。
KeyExists(Key)	キーが既知 (アクティブ、失効済み、または期限切れ) の場合に True を返します。
GrantScope(Key, Scope)	既存のキーにスコープを追加します。
RevokeScope(Key, Scope)	既存のキーからスコープを削除します。
HasScope(Key, Scope)	キーが指定されたスコープを持っている場合は True を返します。
GetAuditLog(Key, MaxEntries)	監査エントリを返します。Key が空の場合、すべてのキーのエントリを返します。
ClearAuditLog(Key)	特定のキーまたは (空の場合) すべてのキーの監査ログをクリアします。
SaveToFile(FileName)	指定したファイルにキーと監査ログを永続化します (空の場合は Storage.FileName にフォールバックします)。
LoadFromFile(FileName)	ファイルからキーと監査ログを復元します。
ExportKeys(Stream)	キーと監査ログのストリームベースエクスポート。
ImportKeys(Stream)	キーと監査ログのストリームベースのインポート。
PurgeExpired	ExpiresAtが過去であるすべてのキーを削除します。
Count	現在保存されているキーの数を返します。
IsRequestAuthorized(Headers, QueryString, IP, RequiredScope)	ワンショット認証チェック：ヘッダーまたはクエリ文字列からキーを抽出し、検証してオプションでスコープを適用します。

ExtractKeyFromHeaders(Headers)	生の HTTP ヘッダーブロックから Validation.HeaderName の値を解析します。
ExtractKeyFromQuery(QueryString)	生のクエリ文字列から Validation.QueryParamName の値を解析します。
IsConnectionAllowed(IP)	サーバーフック: FailClosedがTrueで、IPAllowlistが空でなく、IP がリストにない場合にFalseを返します。自動的に呼び出されます。
IsMessageAllowed(IP, Message)	すべての受信メッセージに対して自動的に呼び出されるサーバーフック。デフォルトでは IP レベルの操作なし。
RegisterConnection(IP)	新しい接続を追跡します。サーバーによって自動的に呼び出されます。
UnregisterConnection(IP)	接続の追跡を解放します。切断時に自動的に呼び出されます。

自動統合

サーバーの APIKeyManager プロパティに割り当てられると、コンポーネントは自動的に以下を行います。

- **新しい接続時:** サーバーは IsConnectionAllowed(IP) を呼び出し、FailClosed が True の場合にオプションの IPAllowlist を適用します（それ以外の場合はノーオペレーション）。RegisterConnection(IP) を呼び出してアクティブな接続を追跡します。
- **メッセージ受信時:** サーバーはIsMessageAllowed(IP, Message)を呼び出します。これも、許可リストがIPを拒否しない限り、IPレベルのno-opです。これらのフックはHTTPヘッダーを検査しないため、APIキー認証はここでは実行されません。
- **認可をリクエスト（ユーザーが配線する必要があります）:** サーバーのOnConnectイベントから *IsRequestAuthorized(Connection.HeadersRequest.Text, Connection.URL, Connection.IP, RequiredScope)* を呼び出します。これは生のHTTPヘッダーが利用可能なポイントであるため、ここでX-API-Keyヘッダーが抽出、検証され、オプションでスコープチェックされます。関数がFalseを返した場合は接続を切断してください。
- **クライアント側:** サーバーがHTTPアップグレード中に読み取れるように、クライアントのOnHandshake / OnBeforeConnectイベントからX-API-Keyヘッダー（または構成されたHeaderName）を設定します。
- **切断時:** UnregisterConnection(IP) が呼び出されて内部追跡が解放されます。

完全な使用例

```
// 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");
```

スレッドセーフ

APIキーマネージャーは完全にスレッドセーフです。すべてのパブリックメソッドは内部クリティカルセクションとスレッドセーフなリストを使用して、キーストア、スコープインデックス、オーナーインデックス、監査ログへの並行アクセスを保護します。単一のコンポーネントを複数のサーバーインスタンスで共有し、外部同期なしに任意のスレッド（サーバーイベントハンドラー、タイマースレッド、バックグラウンドワーカー）からアクセスできます。

リファレンス

- プロパティ — コンポーネントで使用可能な 10 のプロパティ。
- メソッド — 26 個のパブリックメソッド。
- イベント — 処理できる 7 つのイベント。

TsgcWebSocketLoadBalancerServer

TsgcWebSocketLoadBalancerServer コンポーネント。

はじめに

コンポーネント **TsgcWebSocketLoadBalancerServer** は **WebSocket** と **HTTP** プロトコルの負荷分散を可能にします。WebSocket プロトコルの場合、サーバーのグループ間でメッセージを分散し、ランダムシーケンスまたは最少接続数アルゴリズムを使用してクライアント接続を分散します。

ロードバランサーサーバーは **TsgcWebSocketHTTPServer** からすべてのメソッドとプロパティを継承します。

ロードバランサーの設定

Load Balancer サーバーは **TsgcWebSocketHTTPServer** の子孫なので、その構成方法を知るには **TsgcWebSocketHTTPServer** のドキュメントをお読みください。

さらに、ロードバランサーには LoadBalancer プロパティがあり、以下のプロパティが含まれています。

- **LoadBalancing**：接続の分配方法を設定します。
 - **lbRandom**：（デフォルト）新しいクライアントが接続をリクエストするたびに、ランダムなサーバーを返します。
 - **lbConnections**：新しいクライアントが接続をリクエストするたびに、接続クライアントが最も少ないサーバーを返します。
- プロトコル：有効なプロトコルを設定します。
 - **WebSocket**: true の場合、WebSocket 接続はロードバランサーサーバーによって処理されます。
 - **HTTP**: true の場合、HTTP 接続はロードバランサーサーバーによって処理されます。

バックアップサーバーの設定

バックアップサーバー（ロードバランサーの背後にあるサーバー）には、**TsgcWebSocketServer**、**TsgcWebSocketHTTPServer**、または **Datasnap サーバー** を使用できます。

これらのサーバーには **LoadBalancer** プロパティがあり、ロードバランサーサーバーとバックアップサーバー間の接続を設定できます。

- **Enabled**: バックアップサーバーとして使用する場合は true に設定します。
- **Host**: ロードバランサーが配置されているホスト。
- **Port**: LoadBalancerのリッスンポート。
- **Guid**: このサーバーを識別する一意のID。
- **Bindings**: 接続が転送される先のパブリックアドレスです。例: バックアップWebSocketサーバーがポート 8000でリッスンし、IPアドレスが1.1.1.1の場合は、次のように使用します: ws://1.1.1.1:8000;
- **AutoRegisterBindings**: 有効にすると、LoadBalancerサーバーはバックアップサーバーのBindingsプロパティを使用してパブリックバインディングを設定します。
- **AutoRestart**: 秒単位で、ゼロより大きい場合、バックアップサーバーのロードバランサークライアントは内部watchdogを有効にし、x秒ごとに接続が生きているかどうかをチェックします。閉じている場合は、再接続を試みます。

リファレンス

- プロパティ — コンポーネントで利用可能な18個のプロパティ。
- メソッド — 9個のパブリックメソッド。
- イベント — 処理可能な22のイベント。

TsgcWebSocketProxyServer

TsgcWebSocketProxyServer は、クライアントの *WebSocket* 接続をリッスンし、通常の *TCP/IP* サーバーへデータ接続を転送する *WebSocket* サーバーコンポーネントを実装します。

はじめに

TsgcWebSocketProxyServer は *WebSocket* サーバーコンポーネントを実装します。このコンポーネントはクライアントの *WebSocket* 接続をリッスンし、通常の *TCP/IP* サーバーへのデータ接続を転送します。ブラウザが任意のサーバーに仮想的に接続できるため、ブラウザ接続に特に便利です。

リファレンス

- プロパティ — コンポーネントで使用可能な19のプロパティ。
- メソッド — 公開メソッド 7 件。
- イベント — 処理できる22個のイベント。

TsgclWWebSocketClient

TsgclWWebSocketClient は *WebSocket* サーバーに接続できる *IntraWeb WebSocket* コンポーネントを実装します。

はじめに

TsgclWWebSocketClient は *IntraWeb WebSocket* コンポーネントを実装し、*WebSocket* サーバーに接続できます。このコンポーネントを設定するには、以下の手順に従ってください:

1. フォームに *TsgclWWebSocketClient* コンポーネントをドロップします
2. Host と Port (デフォルトは 80) を設定して、利用可能な *WebSocket* サーバーに接続します。URL プロパティを設定することで、Host、Port、パラメータが URL から自動的に更新されます。例: `wss://127.0.0.1:8080/ws/` の場合の結果:

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

3. TLS (セキュア接続) を使用するかどうかを選択できます。デフォルトでは有効になっていません。

4. 許可するトランスポートを設定します。

WebSockets: 標準 *WebSocket* 実装を使用します。

エミュレーション: ブラウザが *WebSocket* をサポートしていない場合、ループAJAXコールバック接続を使用します

6. 非同期プロシージャを作成し、プロパティ `Active := True` を設定します

Methods

Open: *WebSocket* 接続を開きます。

Close: *WebSocket* 接続を閉じます。

WriteData: sends a message to *WebSocket Server*.

Properties

Connected: 読み取り専用変数で、接続がアクティブな場合にTrueを返し、そうでなければFalseを返します。

JSOpen: 接続が開かれたときにクライアント側で JavaScript コードを含めることができます。

JSClose: 接続が閉じられたときに、クライアント側で実行するJavaScriptコードをここに含めることができます。

JSMessage: クライアントがサーバーからメッセージを受信したときにクライアント側で JavaScript コードを含めることができます。JavaScript 変数 "text" を使用してメッセージ文字列を取得できます。

JSError: エラーが発生したときに、クライアント側に JavaScript コードを含めることができます。JavaScript 変数 "text" を使用してエラーメッセージを取得できます。

リファレンス

- プロパティ — コンポーネントで利用可能な11のプロパティ。
- メソッド — 4つのパブリックメソッド。
- イベント — 処理可能な5個のイベント。

TsgcWSConnection

TsgcWSConnection はクライアント WebSocket 接続のラッパーです。このオブジェクトはサーバーまたはクライアントのイベントでアクセスできます。

メソッド

WriteData: クライアントにメッセージを送信します。

Close: 相手ピアにクローズメッセージを送信します。オプションで「CloseCode」を指定できます。デフォルトでは、送信される値は NORMAL クローズコードです。負のクローズコードを送信した場合、クローズ理由は送信されません。

Disconnect: サーバー側からクライアント接続を閉じます。オプションで「CloseCode」を指定できます。

Ping: クライアントに ping を送信します。

AddTCPEndOfFrame: 接続がプレーン TCP の場合、メッセージの終端を定義するバイトを設定できます。メッセージは完全に受信されるまでバッファリングされます。

Subscribed: 接続がカスタムチャンネルをサブスクライブしているかどうかを返します。

Subscribe: この接続をチャンネルにサブスクライブします。後で、サーバーコンポーネントから、このチャンネルにサブスクライブしているすべての接続へメッセージをブロードキャストできます。

UnSubscribe: この接続のチャンネルのサブスクライブを解除します。

プロパティ

Protocol: この接続で使用されるサブプロトコルを返します。

IP: ピア IP アドレスを返します。

Port: ピアポートを返します。

LocalIP: ホストの IP アドレスを返します。

LocalPort: ホストポートを返します。

URL: クライアントによってリクエストされた URL を返します。

Guid: 接続 ID を返します。

HeadersRequest: リクエストで受信したヘッダーのリストを返します。

HeadersResponse: レスポンスとして送信されたヘッダーのリストを返します。

RecBytes: 受信バイト数。

SendBytes: 送信されたバイト数。

Transport: 接続のトランスポートタイプを返します:

trpRFC6455: 通常のWebSocket接続。

trpHixie76: ドラフトWebSocket仕様を使用するWebSocket接続。

trpFlash: Flashをフォールバックとして使用するWebSocket接続。

trpSSE: Server-Sent Events 接続。

trpTCP: プレーン TCP 接続。

TCPEndOfFrameScanBuffer: メッセージの終端を見つける方法を定義できます（トランスポートとして trpTCP を使用する場合）。

eofScanNone: 新しいパケットが到着するたびに OnBinary イベントが呼び出されます。

eofScanLatestBytes: 最新のバイトが AddTCPEndOfFrame メソッドで追加されたバイトと等しい場合、OnBinary メッセージが呼び出されます。そうでない場合、このパケットはバッファリングされます。

eofScanAllBytes: AddTCPEndOfFrame メソッドで追加されたバイトと等しいバイトをパケット全体で検索します。見つかった場合は OnBinary イベントが呼び出され、見つからない場合はパケットがバッファリングされます。

Data: ユーザーセッションデータオブジェクト。ここでオブジェクトを渡し、必要なときにいつでもアクセスできます。例えば、データベースへの接続やユーザーセッションプロパティを渡すことができます。

Protocols

WebSocketを使用すると、サブプロトコルを実装してカスタマイズされた通信を作成できます。例えば、WebSocketプロトコル上にサブプロトコルを実装してJSONメッセージを使用するカスタムアプリケーションと通信したり、XMLメッセージを使用する別のサブプロトコルを実装したりできます。

サーバー側で接続が開かれると、クライアントが送信したサブプロトコルがサーバーでサポートされているかどうかを検証します。サポートされていない場合は接続を閉じます。サーバーは複数のサブプロトコルを実装できますが、1つの接続では1つのみ使用できます。

サブプロトコルは、カスタマイズされたアプリケーションを作成し、すべてのクライアントが同じ通信インターフェースをサポートすることを確保するのに非常に役立ちます。

プロトコル名は任意ですが、"dataset.esegce.com"のような一意の名前を使用することが推奨されます

sgcWebSockets パッケージを使用すると、独自のプロトコルを構築でき、また提供されている組み込みサブプロトコルも使用できます。

1. **プロトコル MQTT:** MQTT はクライアントサーバー型のパブリッシュ/サブスクライブメッセージングトランスポートプロトコルです。軽量でオープン、シンプルで実装が容易になるよう設計されています。
2. **Protocol AppRTC:** は Google と Mozilla が開発した webrtc デモアプリケーションで、WebRTC API を使用して両方のブラウザが互いに「通信」できるようにします。
3. **プロトコル WebRTC:** ウェブにリアルタイム通信（RTC）機能を提供することを目的としたオープンソースプロジェクト。
4. **プロトコルファイル:** バイナリメッセージを使用して実装され、パケットサイズ、認証、QoS、メッセージ確認応答などを含むファイル送信のサポートを提供します。
5. **Protocol SGC:** JSON-RPC 2.0 メッセージを使用して実装され、RPC、PubSub、トランザクションメッセージ、メッセージ確認応答などのパターンを提供します。
6. **Protocol Dataset :** デフォルトプロトコルから継承し、データセットの変更（新しいレコード、レコードの保存、またはレコードの削除）をサーバーからクライアントに送信できます。
7. **Protocol Presence:** チャットルーム、ドキュメントのコラボレーター、同じ Web ページを閲覧している人、ゲームの競合相手など、チャンネルにサブスクライブしているユーザーを把握できます。
8. **プロトコル WAMP 1.0:** RPC と PubSub という2つの非同期メッセージングパターンを提供するオープンな WebSocket サブプロトコル。
9. **Protocol WAMP 2.0:** RPCとPubSubという2つの非同期メッセージングパターンを提供するオープンな WebSocketサブプロトコルです。
10. **Protocol STOMP:** STOMPは、Simple (またはStreaming) Text Orientated Messaging Protocolです。STOMPは相互運用可能なワイヤフォーマットを提供し、STOMPクライアントが任意のSTOMPメッセージブローカーと通信できるようにすることで、多くの言語、プラットフォーム、ブローカー間で容易かつ広範なメッセージング相互運用性を提供します。
 - 10.1 **STOMP for RabbitMQ :** RabbitMQ ブローカー用クライアント。
 - 10.2 **STOMP for ActiveMQ:** ActiveMQブローカー用のクライアント。
11. **Protocol AMQP :** Advanced Message Queuing Protocol (AMQP 0.9.1) は、メッセージブローカーのベンダーや使用するプラットフォームに関係なく、システム間のメッセージング相互運用性を可能にするオープン標準プロトコルとして作成されました。
12. **プロトコル AMQP1:** Advanced Message Queuing Protocol (AMQP 1.0.0) は、メッセージブローカーのベンダーやプラットフォームに関係なく、システム間のメッセージングの相互運用性を可能にするオープン標準プロトコルとして作成されました。

13. Protocol E2EE： WebSocket 接続を通じて送信されるメッセージは、アプリケーションレベルでエンドツーエンド暗号化されるため、WebSocket サーバーがデータを中継しても、通信するクライアントのみが読み取ることができます。

単一接続で複数のプロトコルを使用する必要がある場合（例: リモートプロシージャコールを処理するためにデフォルトプロトコルを、データベース接続を処理するためにデータセットプロトコルを使用する必要がある場合）、各プロトコルコンポーネントに「ブローカー」を割り当てることができ、すべてのメッセージはこの仲介プロトコルを使用して交換されます（簡単な例については「チケットデモ」を参照してください）。

プロトコルは**実行時**に登録できます。**RegisterProtocol**メソッドを呼び出し、プロトコルコンポーネントをパラメータとして渡すだけです。

Javascript リファレンス

sgcWebSockets で使用されている共通 JavaScript ライブラリについての詳細は[こちら](#)を参照してください。

プロトコル Javascript

デフォルトでは、sgcWebSockets は **sgcWebSocket.js** ファイルを使用します。

利用可能なメソッドは以下に記載されています。{%host%}と{%port%}変数を必要に応じて置き換えてください。
例: sgcWebSocket サーバーを www.example.com のポート 80 でリッスンするように設定している場合、sgcWebSocket.js ファイルへのアクセスを次のように設定する必要があります。

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
```

接続を開く

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
  var socket = new sgcWebSocket('ws://{%host%}:{%port%}');
</script>
```

sgcWebSocket には 3 つのパラメータがあり、最初のもののみ 必須です:

sgcWebSocket(url, protocol, transport)

- **URL:** WebSocket サーバーの場所。通常の WebSocket 接続には "ws:" を使用し、セキュアな WebSocket 接続には "wss:" を使用できます。

```
sgcWebSocket('ws://127.0.0.1')
```

```
sgcWebSocket('wss://127.0.0.1')
```

- **Protocol:** サーバーが 1 つ以上のプロトコルを受け入れる場合、使用したいプロトコルを定義できます。

```
sgcWebSocket('ws://127.0.0.1', 'esegece.com')
```

- **Transport:** デフォルトでは、最初に WebSocket 接続を試み、ブラウザで実装されていない場合は Server Sent Events をトランスポートとして使用します。

WebSocket が実装されている場合はそれを使用し、そうでない場合は Server Sent Events を使用します。

```
sgcWebSocket('ws://127.0.0.1')
```

WebSocket のみをトランスポートとして使用する:

```
sgcWebSocket('ws://127.0.0.1', '', ['websocket'])
```

トランスポートとして Server Sent Events のみを使用します:

```
sgcWebSocket('ws://127.0.0.1', '', ['sse'])
```

認証付き接続のオープン

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
```



```
var socket = new sgcWebSocket({"host":"ws://{%host%}:{%port%}", "user":"admin", "password":"1234"});
</script>
```

Send Message

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
  var socket = new sgcWebSocket('ws://{%host%}:{%port%}');
  socket.send('Hello sgcWebSockets!');
</script>
```

受信メッセージのアラートを表示

```
<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>
```

バイナリメッセージの受信

```
<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>
```

バイナリ（ヘッダー + 画像）メッセージを受信しました

```
<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イベントのアラート表示

```
<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>
```

接続を閉じる

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
  socket.close();
</script>
```

接続ステータスを取得する

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script>
  socket.state();
</script>
```

プロトコル MQTT

MQTTはクライアント・サーバー型のパブリッシュ/サブスクライブメッセージングトランスポートプロトコルです。軽量、オープン、シンプルで実装しやすく設計されています。これらの特性により、小さなコードフットプリントが必要であったり、ネットワーク帯域幅が限られていたりする、Machine to Machine (M2M) やInternet of Things (IoT) コンテキストでの通信など、多くの制約のある環境を含む様々な状況での使用に理想的です。

このプロトコルはTCP/IP上、または順序付けられた損失のない双方向接続を提供する他のネットワークプロトコル上で実行されます。機能には以下が含まれます：

- ・ 一対多のメッセージ配信とアプリケーションの分離を提供する、publish/subscribe メッセージパターンを使用します。
- ・ ペイロードのコンテンツに依存しないメッセージングトランスポート。
- ・ メッセージ配信の サービス品質レベルが 3 種類あります：
 - ・ 「最大 1 回」。メッセージはオペレーティング環境のベストエフォートに従って配信されます。メッセージが失われる可能性があります。このレベルは、例えば環境センサーデータなど、次のデータがすぐに公開されるため個々の読み取りが失われても問題ない場合に使用できます。
 - ・ 「少なくとも 1 回」（メッセージの到達は保証されますが、重複が発生する場合があります）。
 - ・ 「正確に1回」。メッセージが確実に1回だけ届くことが保証されます。このレベルは、例えば重複や消失したメッセージが誤った課金につながる可能性がある課金システムなどで使用できます。
- ・ トランスポートオーバーヘッドが小さく、ネットワークトラフィックを削減するためにプロトコル交換が最小化されています。
- ・ 異常な切断が発生したときに関係者に通知するメカニズム。

機能

- ・ MQTT バージョン **3.1.1** および **5.0** をサポートします。
- ・ パブリッシュ/サブスクライブメッセージパターンにより、1対多のメッセージ配信とアプリケーションの分離を実現します。
- ・ **Acknowledgment** 送信されたメッセージの。
- ・ メッセージ配信の **QoS**（サービス品質）を実装します（すべてのレベル: 最大 1 回、最低 1 回、正確に 1 回）
- ・ **Last Will Testament**。
- ・ **セキュア接続**。
- ・ **HeartBeat** および **Watchdog**。
- ・ **認証**（サーバーへ）。

コンポーネント

TsgcWSPClient_MQTT: MQTT Clientコンポーネント。

最も一般的な使用方法

- ・ **接続**
 - ・ [クライアント MQTT 接続](#)
 - ・ [Mosquitto MQTT サーバーへの接続](#)
 - ・ [クライアント MQTT セッション](#)
 - ・ [クライアント MQTT バージョン](#)
- ・ **Publish & Subscribe**
 - ・ [MQTT パブリッシュ サブスクライブ](#)
 - ・ [MQTT トピック](#)
 - ・ [MQTT サブスクライブ](#)

- MQTT メッセージのパブリッシュ
- MQTT メッセージの受信
- MQTT 発行とレスポンスの待機
- その他
 - MQTT 保持メッセージのクリア

TsgcWSPClient_MQTT

MQTT コンポーネントは、バージョン 3 をサポートする軽量でフル機能の MQTT クライアント実装を提供します。

はじめに

MQTT コンポーネントは、バージョン 3.1.1 および 5.0 をサポートする軽量でフル機能の MQTT クライアント実装を提供します。このコンポーネントは、標準 TCP と WebSocket の両方で平文および安全な接続をサポートします。

MQTT サーバーへの接続は簡単です。フォームにこのコンポーネントをドロップし、Client プロパティを使用して [TsgcWebSocketClient](#) コンポーネントを選択します。TsgcWebSocketClient でホストとポートを設定し、Active := True に設定して接続します。

MQTT v5.0はv3.1.1との後方互換性がありません。明らかに多くの新機能が導入されたため、既存の実装を見直す必要があります。

仕様によると、MQTT v5.0 はコアの大部分を維持しながら、MQTTに多くの新機能を追加します。

- Clean Session フラグの機能は、セッション状態データをより細かく制御するために 2 つのプロパティに分割されています: CleanStart パラメータと新しい SessionExpInterval。
- Server disconnect: 接続が閉じられた理由を示すために、サーバーがDISCONNECTを送信できるようにします。
- すべてのレスポンスパケット (CONNACK、PUBACK、PUBREC、PUBREL、PUBCOMP、SUBACK、UNSUBACK、DISCONNECT) に、操作が成功または失敗した理由を説明する理由コードと理由文字列が含まれるようになりました。
- 強化された認証: 相互認証を含むチャレンジ/レスポンス形式の認証を有効にするメカニズムを提供します。これにより、クライアントとサーバーの両方がサポートしている場合に SASL 形式の認証を使用でき、接続内でクライアントが再認証する機能を含みます。
- Request / Responseパターンは、ResponseTopicの追加によって形式化されます。
- 共有サブスクリプション: サブスクリプションの負荷分散されたコンシューマーを可能にする共有サブスクリプションサポートを追加します。
- トピックエイリアスはクライアントとサーバーの両方で送信でき、帯域幅を節約するために短い数値識別子でトピックフィルターを参照できます。
- サーバーは ConnectionProperties でサポートする機能を通知できます。
- サーバー参照: CONNACK または DISCONNECT 時に使用する代替サーバーをサーバーが指定できます。これはリダイレクトまたはプロビジョニングに使用できます。
- 詳細: メッセージの有効期限、受信最大数と最大パケットサイズ、および Will 遅延間隔がすべてサポートされています。

リファレンス

- プロパティ — コンポーネントで利用可能な 10 個のプロパティ。
- メソッド — 9個のpublicメソッド。
- イベント — 処理可能な 13 個のイベント。

TsgcWSPClient_MQTT | クライアント MQTT 接続

MQTT Serverに接続するには、まず[TsgcWebSocketClient](#)と[TsgcWSPClient_MQTT](#)を作成する必要があります。次に、MQTTコンポーネントをWebSocketクライアントにアタッチする必要があります。

基本的な使い方

WebSocket プロトコルを使用して Mosquitto MQTT サーバーに接続します。接続後にトピック "topic1" にサブスクライブします。

```
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");
}
```

クライアント識別子

MQTT はクライアント接続を識別するためにクライアント識別子が必要です。コンポーネントは自動的にランダムな値を設定しますが、必要に応じて独自のクライアント識別子を設定できます。設定するには、**OnBeforeConnect** イベントを処理し、aClientIdentifier パラメータに値を設定します。

```
void OnMQTTBeforeConnect(TsgcWSConnection *Connection, ref bool aCleanSession,
    ref string aClientIdentifier)
{
    aClientIdentifier = "your client id";
}
```

認証

一部のサーバーでは MQTT 接続を認証するためにユーザー名とパスワードが必要です。サーバーに接続する前に **Authentication** プロパティを使用してユーザー名とパスワードの値を設定してください。

```
oMQTT = new TsgcWSPClient_MQTT();
oMQTT->Authentication->Enabled = true;
oMQTT->Authentication->UserName = "your user";
oMQTT->Authentication->Password = "your password";
```

TsgcWSPClient_MQTT | MQTT Mosquitto への接続

以下のサンプル設定を使用して Mosquitto MQTT サーバーに接続します。

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 プレーン 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 | クライアントMQTTセッション

クリーンスタート

OnMQTTBeforeConnect イベントには **aCleanSession** というパラメーターがあります。このパラメーターの値が **True** の場合、クライアントが**新しいセッションを開始したい**ことを意味します。そのため、サーバーにセッションが保存されている場合は破棄する必要があります。したがって、OnMQTTConnect イベントが発火すると、aSession パラメーターは false になります。このパラメーターの値が **False** で、このクライアント識別子に関連付けられたセッションがある場合、サーバーは既存のセッションの状態でクライアントとの通信を再開する必要があります。

クライアントに**予期しない切断**が発生し、切断された場所で**セッションを回復**したい場合は、OnMQTTBeforeConnect で **aCleanSession = True** および **aClientIdentifier = セッションのクライアント ID** を設定します。

セッション

接続が成功したら、**OnMQTTConnect** イベントで Session パラメータの値を確認します。

Session = true は、セッションが再開されたことを意味します。

Session = false は新しいセッションであることを意味します。

```
void OnMQTTBeforeConnect(TsgcWSConnection *Connection, ref bool aCleanSession,
    ref string aClientIdentifier)
{
    aCleanSession = false;
    aClientIdentifier = "previous client id";
}

void OnMQTTConnect(TsgcWSConnection *Connection, const bool Session, const int ReasonCode,
    const string ReasonName, const TsgcWSMQTTCONNACKProperties *ConnectProperties);
{
    if (Session == true)
    {
        WriteLine("Session resumed");
    }
    else
    {
        WriteLine("New Session");
    }
}
```


TsgcWSPClient_MQTT | クライアント MQTT バージョン

現在、MQTT クライアントは以下の仕様をサポートしています。

- **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プロパティを使用して、MQTT Clientコンポーネントが使用する バージョンを選択できます。

MQTT 3.1.1: TsgcWSPClient_MQTT.Version = mqtt311

MQTT 5.0: sgcWSPClient_MQTT.Version = mqtt5

TsgcWSPClient_MQTT | MQTT パブリッシュ サブスクライブ

パブリッシュ/サブスクライブパターン（pub/sub と呼ばれる）は、従来のクライアント/サーバーアーキテクチャの代替手段を提供します。クライアント/サーバーモデルでは、クライアントはエンドポイントと直接通信します。pub/sub モデルはメッセージを送信するクライアント（パブリッシャー）とメッセージを受信するクライアントまたは複数のクライアント（サブスクライバー）を切り離します。パブリッシャーとサブスクライバーは互いに直接連絡することはありません。実際、互いの存在すら知りません。それらの間の接続は第三のコンポーネント（ブローカー）によって処理されます。ブローカーの役割は、受信したすべてのメッセージをフィルタリングし、サブスクライバーに正しく配信することです。

TsgcWSPClient_MQTTを使用してメッセージをパブリッシュしたりトピックにサブスクライブしたりできます。

トピックの購読

接続が成功した後、トピック「topic1」にサブスクライブします。

```
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");
}
```

メッセージのパブリッシュ

「topic1」のすべてのサブスクライバーにメッセージをパブリッシュします

```
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 トピック

トピック

MQTT では、トピックとはブローカーが各接続クライアントのメッセージをフィルタリングするために使用する UTF-8 文字列を指します。トピックは 1 つ以上のトピックレベルで構成されます。各トピックレベルはスラッシュ（トピックレベルセパレーター）で区切られます。

```
myHome / groundfloor / livingroom / temperature
```

メッセージキューと比較して、MQTT トピックは非常に軽量です。クライアントはパブリッシュやサブスクライブの前に目的のトピックを作成する必要がありません。ブローカーは事前の初期化なしに各有効なトピックを受け入れます。各トピックには少なくとも1文字が含まれている必要があり、トピック文字列には空白が許可されません。トピックは大文字と小文字を区別します。

ワイルドカード

クライアントがトピックをサブスクライブするとき、公開されたメッセージの正確なトピックをサブスクライブするか、ワイルドカードを使用して複数のトピックに同時にサブスクライブできます。ワイルドカードはトピックのサブスクライブにのみ使用でき、メッセージの公開には使用できません。2 種類のワイルドカードがあります: `_`単一レベルと `#`マルチレベル。

シングルレベル: +

名前が示すように、シングルレベルのワイルドカードはトピックレベルを 1 つ置き換えます。プラス記号はトピックのシングルレベルのワイルドカードを表します。

```
myHome / groundfloor / + / temperature
```

ワイルドカード付きの単一レベルのサブスクリプションは、ワイルドカードの代わりに任意の文字列を含む任意のトピックと一致します。例えば `_myhome/groundfloor/+/temperature` へのサブスクリプションは次の結果を生成できます:

```
YES => myHome / groundfloor / livingroom / temperature
YES => myHome / groundfloor / kitchen / temperature
NO => myHome / groundfloor / livingroom / brightness
NO => myHome / firstfloor / livingroom / temperature
NO => myHome / groundfloor / kitchen / fridge / temperature
```

マルチレベル:

マルチレベルのワイルドカードは多くのトピックレベルをカバーします。ハッシュ記号はトピックのマルチレベルのワイルドカードを表します。ブローカーがどのトピックが一致するかを判断するために、マルチレベルのワイルドカードはトピックの最後の文字として配置され、スラッシュの後に続く必要があります。

```
myHome / groundfloor / #
```

```
YES => myHome / groundfloor / livingroom / temperature
YES => myHome / groundfloor / kitchen / temperature
YES => myHome / groundfloor / kitchen / brightness
NO => myHome / firstfloor / kitchen / temperature
```

クライアントがマルチレベルワイルドカードを使用してトピックにサブスクライブすると、ワイルドカード文字の前のパターンで始まるすべてのトピックのメッセージを、トピックの長さや深さに関係なく受信します。マルチレ

ベルワイルドカードのみをトピックとして指定した場合（_#）、MQTTブローカーに送信されるすべてのメッセージを受信します。

TsgcWSPClient_MQTT | MQTT Subscribe

TsgcWSPClient_MQTT の Subscribe メソッドを使用してトピックをサブスクライブできます。このメソッドには以下のパラメータがあります：

Topic：サブスクライブするトピックの名前。

QoS：3 つの QoS レベルのうちの1つ（すべてのブローカーがすべての 3 レベルをサポートしているわけではありません）。指定しない場合、mtqsAtMostOnce を使用します。[QoS レベル](#)について詳しく読む。

SubscribeProperties：MQTT 5.0 の場合、サブスクリプションに関する追加プロパティです。

Subscribe QoS = At Least Once

```
MQTT->Subscribe("topic1", mtqsAtLeastOnce);
```

MQTT 5.0 のサブスクライブ

```
oProperties = new TsgcWSMQTTSubscribe_Properties();  
oProperties->SubscriptionIdentifier = 1234;  
oProperties->UserProperties->Add("name=value");  
  
MQTT->Subscribe("topic1", mtqsAtMostOnce, oProperties);
```

TsgcWSPClient_MQTT | MQTT パブリッシュメッセージ

以下のパラメータを持つ **Publish** メソッドを使用して、トピックのすべてのサブスクライバーにメッセージを公開できます:

Topic: メッセージがパブリッシュされるトピックの名前。

Text: メッセージのテキスト。

QoS: 3 つの QoS レベルのうちの 1 つ (すべてのブローカーがすべての 3 レベルをサポートしているわけではありません)。指定しない場合、`mtqsAtMostOnce` を使用します。[QoS レベル](#)について詳しく読む。

Retain: `true` の場合、このメッセージは保持されます。新しいクライアントがこのトピックにサブスクライブするたびに、このメッセージがそのクライアントに送信されます。

PublishProperties: MQTT 5.0 の場合、これらはメッセージのプロパティです。

シンプルなメッセージを発行します

```
MQTT->Publish("topic1", "Hello Subscribers topic1");
```

Publish QoS = 少なくとも 1 回

```
MQTT->Publish("topic1", "Hello Subscribers topic1", mtqsAtLeastOnce);
```

保持メッセージのパブリッシュ

```
MQTT->Publish("topic1", "Hello Subscribers topic1", mtqsAtMostOnce, true);
```

TsgcWSPClient_MQTT | MQTT メッセージの受信

サーバーから送信されたメッセージは **OnMQTTPublish** イベントで受信されます。このイベントには以下のパラメーターがあります:

Topic: このメッセージに関連付けられたトピックの名前。

Text: メッセージのテキスト。

PublishProperties: MQTT 5.0 の場合、パブリッシュされたメッセージのプロパティです。

パブリッシュ済みメッセージの読み取り

```
void OnMQTTPublish(TsgcWSConnection *Connection, string aTopic, string aText,
    TsgcWSMQTTPublishProperties *PublishProperties)
{
    WriteLine("Topic: " + aTopic + ". Message: " + aText);
}
```

TsgcWSPClient_MQTT | MQTT メッセージの受信（拡張）

OnMQTTPublishEx イベントは、**TsgcWSMQTTPublishData** オブジェクトを通じて複数の形式で公開されたメッセージペイロードを提供します。このイベントには以下のパラメーターがあります。

Topic: このメッセージに関連付けられたトピックの名前。

Data: パブリッシュされたメッセージのペイロードを含みます。以下のプロパティがあります:

- **Value:** 文字列としてのペイロード。
- **Bytes:** TBytes としての生のペイロード。
- **Stream:** TMemoryStream としての生のペイロード。

PublishProperties: MQTT 5.0 の場合、パブリッシュされたメッセージのプロパティです。

公開されたメッセージを読む（拡張）

```
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 | 公開と応答待機

MQTTクライアントは複数種類のQoSレベルの使用をサポートしており、各レベルはメッセージが期待通りに処理されたことを確認するための異なるレベルで機能します。

次のQoSレベルがあります。

- **mtqsAtMostOnce:** (デフォルトでは) メッセージは基盤となる TCP/IP ネットワークのベストエフォートに従って配信されます。レスポンスは期待されず、プロトコルではリトライのセマンティクスは定義されていません。メッセージはサーバーに1回届くか、まったく届かないかのいずれかです。
- **mtqsAtLeastOnce:** サーバーによる メッセージの受信は、ACKNOWLEDGMENTメッセージによって確認応答されます。通信リンク、送信デバイス、または確認応答メッセージのいずれかに識別された障害があり、指定された期間後に 確認応答メッセージが受信されない場合、送信者はメッセージを再送します。メッセージは少なくとも1回サーバーに到着します。QoS レベル1のメッセージには、メッセージにIDパラメータがあります。
- **mtqsExactlyOnce:** メッセージが正確に1回届くことが保証されます。このレベルは例えば、メッセージの重複や損失が誤った請求につながる可能性がある課金システムで使用できます。通信リンクまたは送信デバイスの障害が特定された場合、または指定期間内に確認応答メッセージが受信されない場合、送信者はメッセージを再送します。

OnPubAck または OnPubComp イベントを処理してメッセージがサーバーで処理されたかどうかを確認するか、メソッド **PublishAndWait** を使用してメッセージがサーバーで処理されたかどうかを確認できます。

PublishAndWait の使用法は通常の Publish メソッドと同じですが、Timeout という新しいパラメータがあります。一定の時間が経過してもサーバーからの応答がない場合、メソッドは false を返します。デフォルトでは 10 秒です。

```
if mqtt->PublishAndWait("topic", "text")
{
    ShowMessage("Message processed")
}
else
{
    ShowMessage("Message error");
}
```


TsgcWSPClient_MQTT | MQTT 保持メッセージのクリア

デフォルトでは、すべてのMQTTトピックに保持メッセージを持たせることができます。保持メッセージをクリーンアップするための標準的なMQTTメカニズムは、トピックに空のペイロードを持つ保持メッセージを送信することです。これにより保持メッセージが削除されます。

```
MQTT->Publish("topic1", "", mqttAtMostOnce, true);
```

プロトコル AMQP 0.9.1

Advanced Message Queuing Protocol (AMQP) は、メッセージ指向ミドルウェアのためのオープンスタンダードアプリケーション層プロトコルです。AMQP の定義的な機能は、メッセージ指向、キューイング、ルーティング（ポイントツーポイントおよびパブリッシュサブスクライブを含む）、信頼性、セキュリティです。

AMQPは、幅広い種類のメッセージングアプリケーションと通信パターンを効率的にサポートするように設計されたバイナリアプリケーション層プロトコルです。フロー制御されたメッセージ指向の通信を提供し、最大1回（各メッセージが1回または0回配信される）、最低1回（各メッセージが確実に配信されるが複数回配信される可能性がある）、および正確に1回（メッセージは必ず1回だけ届く）というメッセージ配信の保証を提供します。SASLおよびTLSに基づく認証および/または暗号化も提供します。TCP（伝送制御プロトコル）などの信頼性の高いトランスポート層プロトコルを前提としています。

機能

AMQPは、クライアントとブローカー間で高品質かつ安全なメッセージ配信が必要なあらゆる状況で使用できます。

AMQP は以下の機能を提供します:

- 監視と更新の共有。
- サーバーがリクエストに迅速に応答し、時間のかかるタスクをさらなる処理のために転送することを保証します。
- メッセージを複数の受信者に配布します。
- さらなるデータ取得のためのオフラインクライアントの接続。
- アプリケーションの信頼性とスムーズな動作を向上させます。
- メッセージ配信の信頼性。
- 高速なメッセージ配信。
- メッセージの受け入れ。

コンポーネント

TsgcWSPClient_AMQP: AMQP 0.9.1プロトコルを実装するクライアントコンポーネントです。

最も一般的な使用方法

- **Connection**
 - [Client AMQP Connect](#)
 - [Client AMQP Disconnect](#)
- **コマンド**
 - [AMQP チャンネル](#)
 - [AMQP エクスチェンジ](#)
 - [AMQP キュー](#)
 - [AMQP メッセージのパブリッシュ](#)
 - [AMQP メッセージのコンシューム](#)（非同期）
 - [AMQP メッセージの取得](#)（同期）
 - [AMQP QoS](#)
 - [AMQP トランザクション](#)

TsgcWSPClient_AMQP

AMQP 0.

はじめに

TsgcWSPClient_AMQP クライアントは OASIS 仕様に従った完全な **AMQP 0.9.1** プロトコルを実装しています。クライアントはプレーン TCP と WebSocket 接続をサポートし、TLS (セキュア) 接続もサポートします。

接続

AMQP 0.9.1 プロトコルはチャンネルの概念を定義しています。これにより、複数の仮想チャンネルで単一のソケット接続を共有できます。クライアントは、受信したバイトを読み取り、すべてのメッセージを正しいチャンネルにディスパッチする内部スレッドを実装します (各チャンネルはすでに独自のスレッドで実行されています)。AMQP 接続で 5 つのチャンネルを実行している場合、クライアントは 6 つのスレッドを実行します (各チャンネルのデータを処理する 5 つのスレッドと、接続のデータを処理する 1 つのスレッド)。

AMQPサーバーに接続する前に、AMQPプロトコルの以下のプロパティを設定してください

- **AMQPOptions.Locale:** 使用するメッセージのロケールです。これはネゴシエートされる値であるため、サーバーがサポートするロケールと比較すると変更される可能性があります。デフォルト値は"en_US"です。
- **AMQPOptions.MaxChannels:** 開くことができるチャンネルの最大数。ネゴシエートされた値であるため、サーバー設定と比較して変わる場合があります。デフォルト値は 65535 です。
- **AMQPOptions.MaxFrameSize:** AMQP フレームの最大サイズ (バイト単位) です。ネゴシエートされた値であるため、サーバー設定と比較して変わる場合があります。デフォルト値は 2147483647 です。
- **AMQPOptions.VirtualHost:** 仮想ホストの名前です。デフォルト値は「/」です。

AMQP ハートビートはサーバーへの接続前にも設定でき、ハートビートの使用を有効または無効にできます。

- **HeartBeat.Enabled:** クライアントがハートビートをサポートする場合は True に設定します。
- **HeartBeat.Interval:** 望ましい間隔 (秒単位) です。

AMQP クライアントが設定されたら、[TsgcWebSocketClient](#) に接続し、AMQP サーバーへの接続のためのサーバー接続プロパティを設定できます。

WebSocket 接続の代わりにプレーン TCP 接続を使用する場合は、プロパティ値 **Specifications.RFC6455** を false に設定します。

```
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;
```

チャンネル

AMQPクライアントが接続した後、最初のチャンネルを開くことができます。

```
oAMQP->OpenChannel("channel_name");
```

取引所

チャンネルが開かれると、クライアントは新しいエクスチェンジを宣言したり、存在を確認したりできます。新しいエクスチェンジを宣言するには、メソッド **DeclareExchange** を使用してください。

```
oAMQP->DeclareExchange("channel_name", "exchange_name");
```

キュー

チャンネルが開かれると、クライアントは新しいキューを宣言したり、存在を確認したりできます。新しいキューを宣言するには **DeclareQueue** メソッドを使用します。キューはサーバーによってデフォルトで提供されません（エクスチェンジとは異なり）。そのため、新しいキューを常に宣言する必要があります（別のクライアントがキューを作成済みでない限り）。

```
oAMQP->DeclareQueue("channel_name", "queue_name");
```

キューのバインディング

Exchange とキューが設定されたら、キューを Exchange にバインドする必要があります。これにより、Exchange はどのメッセージがどのキューにディスパッチされるかを把握できます。

AMQP サーバーはキュー名をルーティングキーとして使用してキューを "direct" エクスチェンジに自動的にバインドします。これにより、バインディングを宣言せずに特定のキューにメッセージを送信できます（**PublishMessage** メソッドを呼び出し、Exchange 引数を空の値として渡し、RoutingKey 引数にキューの名前を渡すだけです）。

```
oAMQP->BindQueue("channel_name", "queue_name", "exchange_name", "routing_key");
```

Send Messages

新しいAMQPメッセージを発行するには、**PublishMessage**メソッドを呼び出します。このメソッドでは、**String** または**TStream**メッセージを発行できます。

```
oAMQP->PublishMessage("channel_name", "exchange_name", "routing_key", "Hello from sgcWebSockets!!!");
```

メッセージの受信

AMQP では2つのモードでメッセージを受信できます:

- クライアントによるリクエスト: **GetMessage** メソッドを使用します。キューにメッセージがない場合は、**OnAMQPBasicGetEmpty** イベントが呼び出されます。
- サーバーからプッシュ: **Consume** メソッドを使用します。

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->AsStr  
  }
```

サーバーからプッシュ

```
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  
}
```

リファレンス

- プロパティ — コンポーネントで利用可能な 6 つのプロパティ。
- メソッド — 41 個のパブリックメソッド。
- イベント — 処理可能な 25 個のイベント。

接続 | クライアントAMQP接続

AMQP サーバーに接続するには、まず [TsgcWebSocketClient](#) と [TsgcWSPClient_AMQP](#) を作成する必要があります。次に、AMQP コンポーネントを WebSocket クライアントに接続してください。

基本的な使い方

認証なしで AMQP サーバーに接続します。AMQPOptions プロパティ値、仮想ホストを定義し、TsgcWebSocketClient にサーバーのホストとポートを設定します。TCP プレーン接続を使用している場合は、TsgcWebSocketClient プロパティ Specifications.RFC6455 を false に設定してください。

```
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;
```

認証

サーバーが認証を必要とする場合、**OnAMQPAuthentication** イベントを使用して認証メカニズムを選択し (必要に応じて)、ユーザー名/パスワードを設定してください。

```
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

クライアントは、以下のメソッドを使用して、現在のアクティブな接続を切断できます:

クローズ理由の送信

AMQP クライアントはサーバーに接続を閉じることを通知し、接続を閉じる理由に関する情報を提供できます。サーバーへの接続クローズをリクエストするには `Close` メソッドを使用します。

```
oAMQP.Close(541, "Internal Error");
```

ソケット接続を閉じます

`TsgcWebSocketClient` の `Active` プロパティを `False` に設定するだけです。[接続を閉じる](#) 方法について詳しく読むことができます。

Commands | AMQP Channels

AMQP はマルチチャンネルプロトコルです。チャンネルは重量のある TCP/IP 接続を複数の軽量な接続に多重化する手段を提供します。これにより、ポートの使用が予測可能になるためプロトコルが「ファイアウォールフレンドリー」になります。また、トラフィックシェーピングやその他のネットワーク QoS 機能を簡単に採用することも意味します。

各チャンネルは独自のスレッドで実行されるため、新しいメッセージが受信されるたびに、まずクライアントがチャンネルを識別し、チャンネルスレッドによって処理されるキューにメッセージを格納します。

チャンネルのライフサイクルは次のとおりです:

1. クライアントが新しいチャンネルを開きます (Open)。
2. サーバーが新しいチャンネルの準備完了を確認します (Open-Ok)。
3. クライアントとサーバーはチャンネルを必要に応じて使用します。
4. いずれかのピア (クライアントまたはサーバー) がチャンネルを閉じます (Close)。
5. 相手側がチャンネルのクローズをハンドシェイクします (Close-Ok)。

チャンネルを開く

新しいチャンネルを作成するには **OpenChannel** メソッドを呼び出してチャンネル名を引数として渡すだけです。**OnAMQPChannelOpen** イベントは、チャンネルが開かれたというサーバーからの確認として発生します。

```
AMQP->OpenChannel("channel_name");

private void OnAMQPChannelOpen(TObject *Sender, const string aChannel)
{
    DoLog("#AMQP_channel_open: " + aChannel);
}
```

同期呼び出しは **OpenChannelEx** メソッドを呼び出すことでも実行できます。このメソッドはチャンネルが開かれた場合は true を返し、サーバーからの確認が届かなかった場合は false を返します。

```
if AMQP->OpenChannelEx("channel_name")
{
    DoLog("#AMQP_channel_open channel_name");
}
else
{
    DoLog("#AMQP_channel_open_error");
}
```

チャンネルを閉じる

既存のチャンネルを閉じるには、**CloseChannel** メソッドを呼び出してチャンネル名を引数として渡します。クライアントがチャンネルのクローズ確認を受信すると、**OnAMQPChannelClose** イベントが呼び出されます。

同期呼び出しは **CloseChannelEx** メソッドを呼び出すことで実行できます。このメソッドは、チャンネルが閉じられた場合は true を返し、サーバーからの確認が届いていない場合は false を返します。

Channel Flow

フロー制御は、ピアからのメッセージフローを停止するために使用される緊急手順です。クライアントとサーバー間で同様に機能し、**EnableChannel / DisableChannel** コマンドによって実装されます。フロー制御は、過剰に生産するパブリッシャーを停止できる唯一のメカニズムです。

チャンネルのフローを無効にするには、**DisableChannel** メソッドを呼び出します。クライアントがチャンネルフローが無効になったことの確認を受け取ると、**OnAMQPChannelFlow** イベントが呼び出されます。チャンネルのフローを有効にする場合も同様です。**EnableChannel** メソッドを呼び出すと、クライアントがチャンネルフローが有効になったことの確認を受信したときに **OnAMQPChannelFlow** イベントが呼び出されます。

同期リクエストは **EnableChannelEx** と **DisableChannelEx** 関数を通じて利用できます。

コマンド | AMQP エクスチェンジ

exchangeクラスを使用すると、アプリケーションはサーバー上のexchangeを管理できます。このクラスにより、アプリケーションは（何らかの構成インターフェースに依存するのではなく）独自の配線をスクリプト化できます。注意: ほとんどのアプリケーションはこのレベルの高度な機能を必要とせず、レガシーミドルウェアはこのセマンティクスをサポートできない可能性が高いです。

エクスチェンジのライフサイクルは次のとおりです:

1. クライアントはサーバーに対してエクスチェンジが存在するを確認するよう要求します (Declare)。クライアントはこれを「エクスチェンジが存在しない場合は作成する」または「存在しない場合は警告するが作成しない」に絞り込みます。
2. クライアントはメッセージをエクスチェンジに公開します。
3. クライアントは取引所の削除を選択できます (Delete)。

Exchange の宣言

このメソッドは新しいexchangeを作成するか、Exchangeがすでに存在することを確認します。このメソッドには以下の引数があります:

- **ChannelName:** チャンネルの名前（このメソッドを呼び出す前にオープンしておく必要があります）。
- **ExchangeName:** エクスチェンジの名前で、255文字を超えてはならず、「amq.」で始まってはなりません（passiveパラメータがtrueの場合を除く）。
- **ExchangeType:** エクスチェンジのタイプです。すべての AMQP サーバーは「direct」と「fanout」エクスチェンジをサポートしています。サポートされているエクスチェンジタイプについてはサーバーのドキュメントを確認してください。
- **Passive:** passive が true の場合、サーバーはエクスチェンジがすでに宣言されているかどうかのみを確認します。passive が false で、エクスチェンジが存在しない場合、サーバーは新しいものを作成します。
- **Durable:** true の場合、サーバーが起動するときに Exchange が再作成されます。false の場合、サーバーが停止するときに Exchange が削除されます。
- **AutoDelete:** true の場合、すべてのキューがアンバインドされたときにエクスチェンジが削除されます。
- **Internal:** 常に false。
- **NoWait:** true の場合、サーバーはクライアントに確認応答を送信しません。
- **Arguments:** カスタム引数を含む文字列。値はJSON文字列として渡す必要があります。例: {"x-dead-letter-exchange": "my-dlx"}。

新しいエクスチェンジを宣言するには、**DeclareExchange** メソッドを呼び出して、チャンネル名、エクスチェンジ名、エクスチェンジ タイプを引数として渡します。エクスチェンジが宣言されたことのサーバーからの確認として **OnAMQPExchangeDeclare** イベントが発生します。

```
AMQP->DeclareExchange("channel_name", "exchange_name", "direct");

private void OnAMQPExchangeDeclare(TObject *Sender, const string aChannel, const string aExchange)
{
    DoLog("#AMQP_exchange_declare: [" + aChannel + "] " + aExchange);
}
```

メソッド**DeclareExchangeEx**を呼び出すことで、同期呼び出しも実行できます。このメソッドは、ExchangeがDeclareされた場合にtrueを返し、サーバーからの確認が到着しなかった場合にfalseを返します。

```
if AMQP->DeclareExchangeEx("channel_name", "exchange_name", "direct")
{
    DoLog("#AMQP_exchange_declare: [" + aChannel + "] " + aExchange);
}
else
{
    DoLog("#AMQP_exchange_declare_error");
}
```

Delete Exchange

このメソッドは、既存のExchangeを削除するために使用されます。このメソッドには次の引数があります:

- **ChannelName:** チャンネルの名前（このメソッドを呼び出す前にオープンしておく必要があります）。
- **ExchangeName:** エクスチェンジの名前で、255文字を超えてはならず、「amq.」で始まってはなりません（passiveパラメータがtrueの場合を除く）。
- **IfUnused:** サーバーは、exchangeにバインドされたキューが1つもない場合にのみexchangeを削除します。
- **NoWait:** true の場合、サーバーはクライアントに確認応答を送信しません。

既存のエクスチェンジを削除するには **DeleteExchange** メソッドを呼び出し、チャンネル名とエクスチェンジ名を引数として渡します。**OnAMQPExchangeDelete** イベントは、エクスチェンジが削除されたというサーバーからの確認として発生します。

DeleteExchangeExメソッドを呼び出して、同期呼び出しを行うこともできます。このメソッドは、Exchangeが削除された場合はtrueを、サーバーから確認が到着しなかった場合はfalseを返します。

コマンド | AMQP キュー

キュークラスにより、アプリケーションはサーバー上のメッセージキューを管理できます。これは、期待されるメッセージキューが実際に存在することを確認するために、メッセージを消費するほぼすべてのアプリケーションの基本的なステップです。

永続的メッセージキューのライフサイクルは非常にシンプルです。

1. クライアントはメッセージキューが存在することをアサートします（「passive」引数付きの Declare）。
2. サーバーはメッセージキューが存在することを確認します（Declare-Ok）。
3. クライアントはメッセージキューからメッセージを読み取ります。

一時メッセージキューのライフサイクルはより複雑です。

1. クライアントはメッセージキューを作成します（Declare、多くの場合メッセージキュー名なしでサーバーが名前を割り当てます）。サーバーは確認します（Declare-Ok）。
2. クライアントはメッセージキューでコンシューマーを起動します。コンシューマーの具体的な機能はBasicクラスによって定義されます。
3. クライアントは、明示的に、またはチャンネルおよび/または接続を閉じることによって、コンシューマーをキャンセルします。
4. メッセージキューから最後のコンシューマーが消えた後、一定の猶予タイムアウトを経てサーバーはメッセージキューを削除します。

AMQP はトピックサブスクリプションのデリバリーメカニズムをメッセージキューとして実装します。これにより、サブスクリプションが協力するサブスクライバーのプールに負荷分散できる興味深い構造が可能になりますアプリケーション。

サブスクリプションのライフサイクルには追加のバインドステージが含まれます。

1. クライアントがメッセージキューを作成し（Declare）、サーバーが確認します（Declare-Ok）。
2. クライアントはメッセージキューをトピックエクスチェンジにバインドし（Bind）、サーバーが確認します（Bind-Ok）。
3. クライアントは前の例と同様にメッセージキューを使用します。

キューを宣言する

このメソッドは新しいキューを作成するか、キューがすでに存在することを確認します。このメソッドには以下の引数があります：

- **ChannelName:** チャンネルの名前（このメソッドを呼び出す前にオープンしておく必要があります）。
- **QueueName:** キューの名前。255 文字以内で "amq." で始まってはいけません (passive パラメータが true の場合を除く)。
- **Passive:** passive が true の場合、サーバーはキューが既に宣言されているかどうかのみを確認します。passive が false で、キューが存在しない場合、サーバーは新しいキューを作成します。
- **Durable:** true の場合、サーバーの起動時にキューが再作成されます。false の場合、サーバーの停止時にキューが削除されます。
- **Exclusive:** true の場合、キューは現在の接続のみがアクセスできます。
- **AutoDelete:** true の場合、すべてのコンシューマーがキューを使用しなくなったときにキューが削除されます。
- **NoWait:** true の場合、サーバーはクライアントに確認応答を送信しません。
- **Arguments:** カスタム引数を含む文字列。値はJSON文字列として渡す必要があります。例: {"x-dead-letter-exchange": "my-dlx"}。

新しいキューを宣言するには **DeclareQueue** メソッドを呼び出し、チャンネル名とキュー名を引数として渡してください。 **OnAMQPQueueDeclare** イベントは、取引所が宣言されたというサーバーからの確認として発生します。

```
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));
}
```

メソッド **DeclareQueueEx** を呼び出すことで同期呼び出しも可能です。このメソッドはキューが宣言された場合はtrueを返し、サーバーから確認が届かなかった場合はfalseを返します。

```
if AMQP->DeclareQueueEx("channel_name", "queue_name")
{
    DoLog("#AMQP_queue_declare: [" + aChannel + "] " + aQueue);
}
else
{
    DoLog("#AMQP_queue_declare_error");
}
```

キューの削除

このメソッドは既存のキューを削除するために使用されます。メソッドには次の引数があります:

- **ChannelName:** チャンネルの名前（このメソッドを呼び出す前にオープンしておく必要があります）。
- **QueueName:** キューの名前。255 文字以内で "amq." で始まってはいけません (passive パラメータが true の場合を除く)。
- **IfUnused:** サーバーはコンシューマーがアタッチされていない場合にのみキューを削除します。
- **IfEmpty:** メッセージがない場合にのみサーバーはキューを削除します。
- **NoWait:** true の場合、サーバーはクライアントに確認応答を送信しません。

既存のキューを削除するには **DeleteQueue** メソッドを呼び出し、チャンネル名とキュー名を引数として渡します。 **OnAMQPQueueDelete** イベントは、キューが削除されたというサーバーからの確認として発生します。

DeleteQueueEx メソッドを呼び出すことで同期呼び出しも可能です。このメソッドはキューが削除された場合に true を返し、サーバーから確認が届かなかった場合は false を返します。

キューのバインド

このメソッドはキューをエクスチェンジにバインドするために使用します。エクスチェンジはバインディングを使用して、メッセージのルーティングに使用するキューを決定します。

すべてのAMQP Serverは、Queue Nameをバインディングのルーティングキーとして使用して、すべてのキューを自動的にデフォルトエクスチェンジ（名前のない「direct」エクスチェンジ）にバインドします。これにより、バインディングを宣言せずに特定のキューにメッセージを送信できます。 **PublishMessage**メソッドを呼び出し、Exchange Nameとして空の値を渡し、 **RoutingKey**を **Queue Name**の値に設定するだけです。

このメソッドには以下の引数があります:

- **ChannelName:** チャンネルの名前（このメソッドを呼び出す前にオープンしておく必要があります）。
- **QueueName:** キューの名前。255 文字以内で "amq." で始まってはいけません (passive パラメータが true の場合を除く)。
- **ExchangeName:** エクスチェンジの名前で、255文字を超えてはならず、「amq.」で始まってはなりません (passiveパラメータがtrueの場合を除く)。
- **RoutingKey:** バインディングのルーティングキーです。
- **NoWait:** true の場合、サーバーはクライアントに確認応答を送信しません。
- **Arguments:** カスタム引数を含む文字列。値はJSON文字列として渡す必要があります。例: {"x-dead-letter-exchange":"my-dlx"}。

キューをエクスチェンジにバインドするには、メソッド **BindQueue** を呼び出し、チャンネル名、キュー名、エクスチェンジ、ルーティングキーを引数として渡します。イベント **OnAMQPQueueBind** は、キューがバインドされたことをサーバーが送信する確認として発生します。

```
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 メソッドを呼び出すことで同期呼び出しも可能です。このメソッドはキューがバインドされた場合は **True** を、サーバーからの確認が届かない場合は **False** を返します。

```
if AMQP->BindQueueEx("channel_name", "queue_name", "exchange_name", "routing_key")
{
    DoLog("#AMQP_queue_bind: [" + aChannel + "] " + aQueue + " -->-- " + aExchange);
}
else
{
    DoLog("#AMQP_queue_bind_error");
}
```

キューのバインド解除

このメソッドは既存のキューバインディングを削除します。

このメソッドには以下の引数があります:

- **ChannelName:** チャンネルの名前（このメソッドを呼び出す前にオープンしておく必要があります）。
- **QueueName:** キューの名前。255 文字以内で "amq." で始まってはいけません (passive パラメータが true の場合を除く)。
- **ExchangeName:** エクスチェンジの名前で、255文字を超えてはならず、「amq.」で始まってはなりません (passiveパラメータがtrueの場合を除く)。
- **RoutingKey:** バインディングのルーティングキーです。

キューのバインドを解除するには、**UnBindQueue** メソッドを呼び出し、チャンネル名、キュー名、取引所、ルーティングキーを引数として渡します。**OnAMQPQueueUnBind** イベントは、キューのバインドが解除されたというサーバーからの確認として発生します。

UnBindQueueEx メソッドを呼び出して同期呼び出しも行えます。このメソッドはキューのバインドが解除された場合は **true** を返し、サーバーからの確認が届かなかった場合は **false** を返します。

キューのページ

このメソッドは、キューのすべてのメッセージをページします。送信済みだが確認待ちのすべてのメッセージは影響を受けません。

このメソッドには以下の引数があります:

- **ChannelName:** チャンネルの名前（このメソッドを呼び出す前にオープンしておく必要があります）。
- **QueueName:** キューの名前。255 文字以内で "amq." で始まってはいけません (passive パラメータが true の場合を除く)。
- **NoWait:** true の場合、サーバーはクライアントに確認応答を送信しません。

キューをページするには、**PurgeQueue** メソッドを呼び出し、チャンネル名とキュー名を引数として渡すだけです。キューがページされたことの確認としてサーバーから送信される**OnAMQPQueuePurge**イベントが発生します。

メソッド **PurgeQueueEx** を呼び出すことで同期呼び出しも可能です。このメソッドはキューがページされた場合に **true** を返し、サーバーからの確認が届かない場合は **false** を返します。

コマンド | AMQP メッセージのパブリッシュ

メッセージのパブリッシュ

PublishMessages メソッドは AMQP サーバーにメッセージを送信するために使用します。

AMQP サーバーはキュー名をルーティングキーとして使用してキューを "direct" エクスチェンジに自動的にバインドします。これにより、バインディングを宣言せずに特定のキューにメッセージを送信できます (PublishMessage メソッドを呼び出し、Exchange 引数を空の値として渡し、RoutingKey 引数にキューの名前を渡すだけです)。

このメソッドには以下の引数があります:

- **ChannelName:** チャンネルの名前 (このメソッドを呼び出す前にオープンしておく必要があります)。
- **ExchangeName:** エクスチェンジの名前です。255 文字以内で、「amq.」で始まってはなりません (passive パラメータが true の場合を除く)。
- **RoutingKey:** バインディングのルーティングキー名です。
- **Mandatory:** true の場合、メッセージをキューにルーティングできない場合、メッセージはサーバーによって返され、イベント OnAMQPBasicReturn が発生します。
- **Immediate:** true の場合、メッセージをいずれのキューにもルーティングできないとき、サーバーはメッセージを返し、OnAMQPBasicReturn イベントが発生します。

```
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 + " " +
}
```

公開確認

メッセージの公開中にネットワークが失敗する可能性があります。メッセージが失われないことを保証する唯一の方法はトランザクションを使用することです。各メッセージに対してトランザクションを選択し、メッセージを送信してコミットします。トランザクション成功の確認は OnAMQPTransactionOk イベントが発生したときに受け取られます。

AMQP Consume Messages

コンシューマーはキューから消費します。メッセージを消費するにはキューが必要です。新しいコンシューマーが追加されると、キューにすでにメッセージが準備されている場合、配信は即座に開始されます。コンシューマー登録時にターゲットキューが空場合があります。その場合、新しいメッセージがエンキューされたときに最初の配信が行われます。

メッセージの消費は**非同期**タスクです。これは、コンシューマーキューに新しいメッセージを配信できるたびに、サーバーからクライアントに自動的にプッシュされることを意味します。[同期的にメッセージを受信する](#)代替方法を参照してください。

消費

Consume メソッドはキューに新しいコンシューマーを作成し、新しいメッセージがあるたびにそれがコンシューマークライアントに自動的に配信されます。

このメソッドには以下の引数があります:

- **ChannelName:** チャンネルの名前（このメソッドを呼び出す前にオープンしておく必要があります）。
- **QueueName:** キューの名前。255 文字以内で、"amq." で始まってはなりません（passive パラメータが true の場合を除く）。
- **ConsumerTag:** コンシューマーの名前で、一意である必要があります。設定されていない場合、サーバーが新しい名前を作成します。
- **NoLocal:** true の場合、コンシューマーは同じチャンネルに発行されたメッセージを決して消費しないことを意味します。
- **NoAck:** true の場合、サーバーは配信されたすべてのメッセージに対して確認応答を期待しないことを意味します。
- **Exclusive:** true の場合、他のコンシューマーがこのキューからメッセージを消費することを防ぎます。
- **NoWait:** True の場合、サーバーはクライアントに確認応答を送信しません。
- **Arguments:** カスタム引数を含む文字列。値はJSON文字列として渡す必要があります。例: {"x-dead-letter-exchange": "my-dlx"}。

メッセージは **OnAMQPBasicDeliver** イベントで配信されます。

```
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 メソッドを呼び出すことで同期呼び出しも可能です。このメソッドはコンシューマーが作成された場合に true を返し、サーバーから確認が届かなかった場合は false を返します。

Cancel Consume

このメソッドは、既存のコンシューマーキューをキャンセルするために使用されます。

このメソッドには以下の引数があります:

- **ChannelName:** チャンネルの名前（このメソッドを呼び出す前にオープンしておく必要があります）。
- **ConsumerTag:** コンシューマーの名前です。
- **NoWait:** True の場合、サーバーはクライアントに確認応答を送信しません。


```
AMQP->CancelConsume("channel_name", "consumer_tag");  
  
private void OnAMQPBasicCancelConsume(TObject *Sender, const string aChannel, const string aConsumerTag)  
{  
    DoLog("#AMQP_basic_cancel_consume: " + aChannel + " " + aConsumerTag);  
}
```

メソッド **CancelConsumeEx** を呼び出すだけで同期呼び出しも可能です。このメソッドはコンシューマーがキャンセルされた場合に true を返し、サーバーからの確認が届かない場合は false を返します。

コマンド | AMQP メッセージの取得

メッセージの取得は同期タスクであり、つまりクライアントがキューにメッセージがあるかどうかをサーバーに尋ねます。メッセージを非同期に受信する代替方法を読むことができます。

メッセージを取得する

GetMessage メソッドは、キューにメッセージがあるかどうかを確認するリクエストを AMQP サーバーに送信します。メッセージがある場合は **OnAMQPBasicGetOk** イベントでディスパッチされ、キューが空の場合は **OnAMQPBasicGetEmpty** イベントが呼び出されます。

このメソッドには以下の引数があります:

- **ChannelName:** チャンネルの名前（このメソッドを呼び出す前にオープンしておく必要があります）。
- **QueueName:** キューの名前。255 文字以内で "amq." で始まってはいけません (passive パラメータが true の場合を除く)。
- **NoWait:** True の場合、サーバーはクライアントに確認応答を送信しません。

```
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);
}
```

同期呼び出しは **GetMessageEx** メソッドを呼び出すだけで実行できます。このメソッドはキューにメッセージがある場合は True を返し、それ以外の場合は False を返します。

Commands | AMQP QoS

AMQP では、メッセージの確認応答を待機する前にサーバーがクライアントに送信するメッセージ数を制限するための QoS レベルを設定できます。

QoSの設定

SetQoS メソッドは、サーバーがAMQPクライアントに送信するメッセージ数を制限するために使用します。このメソッドには以下の引数があります:

- **ChannelName:** チャンネルの名前（このメソッドを呼び出す前にオープンしておく必要があります）。
- **PrefetchSize:** ウィンドウサイズ（バイト単位）です。現在未確認の送信済みメッセージの合計サイズと次に送信するメッセージのサイズの合計が **PrefetchSize** 引数を超える場合、サーバーはクライアントにメッセージを送信しません。値がゼロの場合は制限なしを意味します。
- **PrefetchCount:** すでに送信され確認応答されていない未確認メッセージの最大数。数がこれより大きい場合、サーバーはクライアントへのメッセージの送信を停止します。
- **Global:** true の場合、QoS は接続のすべての既存および新しいコンシューマーに適用されます。false の場合、QoS はチャンネルのすべての既存および新しいコンシューマーに適用されます。

サーバーからのレスポンスは **OnAMQPBasicQoS** イベントで受信されます。

```
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));
}
```

同期呼び出しは **SetQoSEx** メソッドを呼び出すだけで実行できます。このメソッドはリクエストが処理された場合に true を返し、そうでない場合は false を返します。

コマンド | AMQP トランザクション

AMQP は 2 種類のトランザクションをサポートしています:

1. 自動トランザクション: すべての発行されたメッセージと確認応答が独立したトランザクションとして処理されます。
2. サーバーローカルトランザクション。サーバーはパブリッシュされたメッセージと確認応答をバッファリングし、クライアントの要求に応じてコミットします。

Transaction クラス (「tx」) はアプリケーションに 2 番目のタイプ (サーバートランザクション) へのアクセスを提供します。このクラスのセマンティクスは次のとおりです:

1. アプリケーションはトランザクションを必要とする各チャンネルでサーバートランザクションを要求します (Select)。
2. アプリケーションが作業を行います (Publish、Ack)。
3. アプリケーションが作業をコミットまたはロールバックします (Commit、Roll-back)。
4. アプリケーションは無限に処理を実行します。

トランザクションは、配信ではなく、発行されたコンテンツと確認応答を対象とします。したがって、ロールバックはメッセージを再キューイングまたは再配信せず、クライアントは 後続のトランザクションでこれらのメッセージを確認応答する権利があります。

トランザクションメソッドにより、パブリッシュおよび確認操作をアトミックな作業単位にバッチ処理できます。トランザクション内で発行されたすべてのパブリッシュおよび確認リクエストはすべて正常に完了するか、すべて失敗することを意図しています。

トランザクションの開始

StartTransaction メソッドはサーバーで新しいトランザクションを開始します。クライアントは Commit または Rollback メソッドを使用する前に、チャンネルでこのメソッドを少なくとも 1 回使用します。サーバーがトランザクションの使用を確認すると、**OnAMQPTransactionOk** イベントが発生します。

```
AMQP->StartTransaction("channel_name");
```

同期呼び出しは、メソッド **StartTransactionEx** を呼び出すだけで実行できます。このメソッドはリクエストが処理された場合は true を返し、そうでない場合は false を返します。

トランザクションのコミット

このメソッドは現在のトランザクションですべてのメッセージのパブリッシュと確認応答をコミットします。コミット後すぐに新しいトランザクションが開始されます。サーバーがトランザクションの使用を確認すると **OnAMQPTransactionOk** イベントが発生します。

```
AMQP->CommitTransaction("channel_name");
```

メソッド **CommitTransactionEx** を呼び出すことで同期呼び出しも可能です。このメソッドはリクエストが処理された場合は true を返し、そうでない場合は false を返します。

トランザクションのロールバック

このメソッドは、現在のトランザクションで実行されたすべてのメッセージパブリケーションと確認応答を破棄します。ロールバック後すぐに新しいトランザクションが開始されます。未確認のメッセージはロールバックによって自動的に再配信されません。それが必要な場合は、明示的なリカバリ呼び出しを実行する必要があります。サーバーがトランザクションの使用を確認すると、イベント **OnAMQPTransactionOk** が発生します。

```
AMQP->RollbackTransaction("channel_name");
```

同期呼び出しは **RollbackTransactionEx** メソッドを呼び出すだけで実行できます。このメソッドはリクエストが処理された場合は `true` を返し、そうでない場合は `false` を返します。

プロトコル Kafka

Apache Kafka は、レコードのストリームをパブリッシュ、保存、消費するために使用されるオープンソースの分散イベントストリーミングプラットフォームです。プロデューサーはトピックにメッセージを書き込み、ブローカーはそれらをパーティション化された追記専用ログに保存し、コンシューマーは個別またはコーディネートされたコンシューマーグループの一部として順番に取り出します。

Kafkaは、TCP上のバイナリのアプリケーション層プロトコルを使用します。各トピックは1つ以上のパーティションに分割され、パーティション内のすべてのレコードは単調増加するオフセットで識別されます。ブローカーはレコードを保持するため、コンシューマーは自分のペースで読み取り、必要に応じて履歴を再生できます。これにより、Kafkaは高スループットのメッセージング、イベントソーシング、ログ集約、ストリーム処理に適しています。

機能

sgcWebSockets Kafka クライアントは、REST プロキシや外部ライブラリを必要とせず、ネイティブ Kafka プロトコルを介してブローカーと直接通信します。

- 任意のトピックとパーティションにキー、値、設定可能な確認応答でレコードをパブリッシュ (プロデュース) します。
- 管理されたコンシューマーグループを使用して、またはグループなしで (すべてのパーティションを直接読み取る) レコードを消費します。
- コンシューマーグループのコーディネーション: コーディネーター検出、参加/同期、パーティション割り当て、オフセットコミット/フェッチ。
- オフセット管理: 最も古い、最新の、およびコミット済みのオフセットをクエリします。
- トピック管理: トピックの作成と削除、クラスターメタデータのリクエスト、コンシューマーグループの一覧表示。
- 現在のKafkaブローカーが使用するv2レコードバッチ形式を読み書きします。

サポートされているバージョン

クライアントは **Apache Kafka 0.11** で導入されたv2レコードバッチ形式を使用し、**Apache Kafka 3.x** に対してテストされています。デフォルトのブローカーポートは9092です。

コンポーネント

TsgcWSPClient_Kafka: Apache Kafka プロトコルを実装するクライアントコンポーネントです。

最も一般的な使用方法

- 接続
 - [クライアント Kafka 接続](#)
 - [クライアント Kafka 切断](#)
- コマンド
 - [Kafka プロデュースメッセージ](#)
 - [Kafka コンシュームメッセージ](#)
 - [Kafka コンシューマーグループ](#)
 - [Kafka トピック](#)
 - [Kafka オフセット](#)

TsgcWSPClient_Kafka

Apache Kafka クライアント。

はじめに

TsgcWSPClient_Kafka クライアントは **Apache Kafka** プロトコルを実装し、プレーンなTCP接続上でネイティブのKafkaバイナリプロトコルを介してブローカーと直接通信します。REST proxyや外部ライブラリなしで、レコードのプロデュースとコンシューム、コンシューマーグループの管理、トピックの管理、オフセットの照会ができます。

接続

Kafka クライアントは **TsgcWebSocketClient** 上で動作するプロトコルコンポーネントです。Kafka は WebSocket プロトコルではないため、**Specifications.RFC6455** を **false** に設定して、クライアントがプレーン TCP ソケットとして接続するようにしてください。次にクライアントを Kafka コンポーネントに割り当て、ブローカーの **Host** と **Port** (デフォルト 9092) を設定してください。

プロデューサーとコンシューマーの動作は **KafkaOptions** プロパティを通じて設定されます:

- **KafkaOptions.ClientId**: ブローカーに対してアプリケーションを識別します。
- **KafkaOptions.Producer.Acks**: 必要な確認応答。kafkaAcksNone、kafkaAcksLeader、または kafkaAcksAll。
- **KafkaOptions.Producer.TimeoutMs**: ブローカーが必要な確認応答を待つ時間。
- **KafkaOptions.Consumer.GroupId**: コンシューマーグループ名です。グループなしで消費するには空のままにします。
- **KafkaOptions.Consumer.OffsetReset**: kafkaOffsetEarliest または kafkaOffsetLatest (新しいグループが読み取りを開始する位置)。

```
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;
```

Produce Messages

レコードを発行するには、**Produce**メソッドを呼び出します。各発行の結果は、レコードが保存されたオフセットを含めて、**OnKafkaProduce**イベントを通じて報告されます。

```
oKafka->KafkaOptions->Producer->Acks = kafkaAcksLeader;
oKafka->Produce("my-topic", "Hello Kafka", "key-1");
```

メッセージの消費

GroupId と **OffsetReset** を設定し、1 つ以上のトピックをサブスクライブして **Poll** を呼び出すと利用可能なレコードを取得できます。Poll は TsgcKafkaMessages リストを返し、各レコードに対して OnKafkaMessage イベントも発生させます。継続的に受信するには、Poll を繰り返し呼び出してください (例: タイマーから)。

```
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();
}
```

リファレンス

- プロパティ — コンポーネントで利用可能なプロパティ3件。
- メソッド — 18個のpublicメソッド。
- イベント — 処理できる 6 つのイベント。

接続 | クライアント Kafka Connect

Apache Kafka ブローカーに接続するには、まず [TsgcWebSocketClient](#) と [TsgcWSPClient_Kafka](#) を作成する必要があります。次に Kafka コンポーネントを WebSocket クライアントにアタッチします。Kafka はプレーン TCP 接続を使用するため、TsgcWebSocketClient プロパティの Specifications.RFC6455 を false に設定する必要があります。ブローカーの Host と Port を設定し (デフォルトの Kafka ポートは 9092)、必要に応じて KafkaOptions.ClientId を設定し、Active を true に設定して接続を開始します。ブローカーへの接続が確立されると、**OnKafkaConnect** イベントが発生します。

基本的な使い方

```
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");
}
```

接続 | クライアント Kafka 切断

Apache Kafka ブローカーから切断するには、WebSocket クライアントプロパティ `Active` を `false` に設定します。これにより、ブローカーへの基礎となる TCP 接続が閉じられます。接続が閉じられると、**OnKafkaDisconnect** イベントが発生します。

基本的な使い方

```
oClient->Active = false;  
  
void OnKafkaDisconnect(TObject *Sender)  
{  
    ShowMessage("Disconnected from Kafka broker");  
}
```

コマンド | Kafka メッセージのプロデュース

Produce(topic, value, key, partition)メソッドを使用して、Kafkaトピックにレコードを発行します。keyとpartitionの引数はオプションです。partitionが設定されていない場合、ブローカーが（たとえばkeyのハッシュ化によって）1つを選択します。バイナリペイロードを発行するには、**ProduceBytes**を呼び出します。

KafkaOptions.Producer.Acks で配信保証を設定します：kafkaAcksNone（送りっぱなし）、kafkaAcksLeader（パーティションリーダーを待つ）、kafkaAcksAll（すべての同期レプリカを待つ）。Producer.TimeoutMs および Producer.Compression（kafkaCompressionNone または kafkaCompressionGzip）も設定できます。

レコードが送信された後、**OnKafkaProduce** イベントは生成されたレコードのトピック、パーティション、オフセット、エラーコードを報告します。

基本的な使い方

```
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));
}
```

コマンド | Kafkaメッセージを消費する

レコードを消費するには、`KafkaOptions.Consumer.GroupId` と `KafkaOptions.Consumer.OffsetReset` (`kafkaOffsetEarliest` または `kafkaOffsetLatest`) を設定し、1つ以上のトピックを指定して **`Subscribe([topic])`** を呼び出します。

サブスクライブ後、**`Poll(timeoutMs)`** を呼び出してレコードを取得します。Poll は `TsgcKafkaMessages` リストを返し、レコードごとに **`OnKafkaMessage`** イベントを発生させます。新しいレコードが到着するたびに消費し続けるため、通常はタイマーなどから繰り返し Poll を呼び出します。

リスト内の各レコードは `TsgcKafkaMessage` です。Count を使用してリストを反復し、`GetKeyString`、`GetValueString`、`Topic`、`Partition`、`Offset` で各レコードを読み取ります。

サブスクライブとポーリング

```
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

Pollが返すリストを反復処理する代わりに、`OnKafkaMessage` イベントで各レコードを処理できます。

```
void OnKafkaMessage(TObject *Sender, TsgcKafkaMessage *Message)
{
    ShowMessage(Message->Topic + ": " + Message->GetKeyString() + " = " + Message->GetValueString());
}
```

コマンド | Kafka コンシューマーグループ

コンシューマーグループを使用すると、複数のコンシューマーがトピックの読み取り作業を共有できます。ブローカーはGroupIdごとにコミットされたオフセットを追跡し、グループのメンバーにトピックパーティションを割り当てるため、各パーティションは一度に1つのメンバーのみが読み取ります。グループはKafkaOptions.Consumer.GroupIdで設定します。

コミット済みオフセットのない新しいグループでは、KafkaOptions.Consumer.OffsetResetが開始位置を決定します。kafkaOffsetEarliestはパーティションの先頭から読み取り、kafkaOffsetLatestは新しいレコードのみを読み取ります。オフセットがコミットされた後は、最後にコミットされた位置から消費が再開されます。

最後のPollで返されたレコードのオフセットをコミットするには**CommitSync**を呼び出します。これにより次のセッションでそれ以降から再開できます。GroupIdを空にした場合、クライアントはグループに参加せずにすべてのパーティションを直接消費し、ブローカーにオフセットをコミットしません。

基本的な使い方

```
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();
```

コマンド | Kafka トピック

Kafka クライアントはブローカー上のトピックを管理できます。**CreateTopic(name, partitions, replication)** を使用して指定されたパーティション数とレプリケーションファクターで新しいトピックを作成し、**DeleteTopic(name)** を使用してトピックを削除します。

GetMetadata([topics])を使用して、ブローカー、トピック、およびそのパーティションレイアウトといったクラスターとトピックの情報を取得します。空の配列を渡すと、クラスター全体のメタデータをリクエストします。

基本的な使い方

```
// 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");
```

コマンド | Kafka オフセット

Kafka クライアントは、指定されたトピックとパーティションのオフセットを照会および設定できます。**GetEarliestOffset(topic, partition)** は利用可能な最も古いオフセットを返し、**GetLatestOffset(topic, partition)** は次に書き込まれるオフセットを返し、**GetCommittedOffset(topic, partition)** はコンシューマーグループによって最後にコミットされたオフセットを返します。3つとも `Int64` を返します。**CommitOffset(topic, partition, offset)** を使用して、コンシューマーグループの特定のオフセットをコミットし、次のセッションでその位置から消費が再開されるようにします。

基本的な使い方

```
__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);
```

プロトコル AMQP 1.0.0

AMQP (Advanced Message Queuing Protocol) 1.0.0 は、分散システム間の信頼性の高い非同期通信のために設計されたメッセージングプロトコルです。アプリケーションまたはコンポーネント間のメッセージ交換を直接的な依存関係なしに可能にし、疎結合な通信を実現します。以下に AMQP 1.0.0 のいくつかの主要な側面の技術的概要を示します。

- **メッセージ指向通信:** AMQP 1.0.0 はメッセージの概念を中心に据えています。メッセージはデータ、指示、またはコマンドを運び、通信の基本単位です。
- **メッセージブローカー:** プロトコルはブローカーメッセージングモデルで動作します。サーバーまたは仲介エンティティであるブローカーは、プロデューサーとコンシューマー間のメッセージのルーティングと配信を管理します。
- **キューとエクスチェンジ:** キューはメッセージが一時的に保存されるブローカー内のストレージエンティティです。エクスチェンジは、メッセージの内容やルーティングキーなどの基準に基づいて、プロデューサーからキューへのメッセージのルーティングルールを定義します。
- **アドレスとリンク:** アドレスはメッセージングインフラストラクチャ内のメッセージの宛先を識別します。リンクは、特定のアドレスに関連付けられた送信者（プロデューサー）と受信者（コンシューマー）間の通信チャンネルです。
- **セッションと接続:** セッションは通信のための論理チャンネルを表し、単一の接続内で複数のメッセージストリームを可能にします。接続はクライアントアプリケーションとメッセージブローカー間の全体的な通信リンクを管理します。
- **セキュリティ:** AMQP 1.0.0 はクライアントとブローカー間の安全な通信を確保するために、認証と認可を含むさまざまなセキュリティメカニズムをサポートしています。
- **トランスポート非依存:** このプロトコルはトランスポート非依存に設計されており、TCP、TLS、WebSocketsなどの異なるネットワークトランスポート上で動作できるため、デプロイメントに柔軟性を提供します。
- **フロー制御:** AMQP 1.0.0 にはフロー制御のメカニズムが含まれており、コンシューマーは特定のレートで受信メッセージを処理する能力を示すことができます。これにより、大量のメッセージでコンシューマーが圧倒されるのを防ぎます。
- **エラー処理:** プロトコルはエラー処理のメカニズムを規定しており、メッセージの確認応答と拒否を含み、メッセージ配信の堅牢性と信頼性を確保します。
- **SASL 認証:** Simple Authentication and Security Layer (SASL) は、クライアントとブローカー間の接続を認証およびセキュリティ保護するために使用されます。

AMQP 1.0.0 は全体として、さまざまなソフトウェアコンポーネントとシステムが疎結合で通信するための標準化された相互運用可能な方法を提供します。これにより、さまざまな分散およびエンタープライズレベルのアプリケーションに適しています。

コンポーネント

TsgcWSPClient_AMQP1: AMQP 1.0.0 プロトコルを実装するクライアントコンポーネントです。

最も一般的な使用方法

- **接続**
 - [クライアント AMQP1 接続](#)
 - [クライアント AMQP1 切断](#)
 - [クライアント AMQP1 アイドルタイムアウト接続](#)
 - [クライアント AMQP1 接続状態](#)
 - [クライアント AMQP1 認証](#)
 - [クライアント AMQP1 Azure Service Bus](#)
- **コマンド**
 - [AMQP1 セッション](#)
 - [AMQP1 リンク](#)
 - [AMQP1 センダーリンク](#)
 - [AMQP1 レシーバーリンク](#)
 - [AMQP1 メッセージ送信](#)
 - [AMQP1 メッセージ読み取り](#)

TsgcWSPClient_AMQP1

AMQP 1.

はじめに

TsgcWSPClient_AMQP1 クライアントは OASIS 仕様に従って **AMQP 1.0.0** プロトコルを実装しています。プレーン TCP と WebSocket 接続をサポートし、TLS（セキュア）接続もサポートしています。

設定

AMQP 1.0.0 クライアントには接続を設定できる **AMQPOptions** プロパティがあります。

- **ChannelMax:** channel-max値は、接続上で使用できる最も大きいチャンネル番号です。この値に1を加えた数が、接続上で同時にアクティブにできるセッションの最大数です
- **ContainerId:** (オプション) ソースコンテナの名前。サーバー内の接続を一意に識別します。
- **CreditSize:** クレジットフローのデフォルトサイズ。
- **IdleTimeout:** このタイムアウトは、しきい値を超えた後にフレームが受信されない場合にローカルピアによってトリガーされます。アイドルタイムアウトはミリ秒単位で測定され、最後のフレームが受信された時点から開始されます。
- **MaxFrameSize:** 受け入れる最大フレームサイズ。
- **MaxLinksPerSession:** セッションあたりの最大リンク数。
- **WindowSize:** デフォルトのウィンドウサイズ。

AMQP認証はAuthenticationプロパティで構成する必要があります。

- **AuthType:** 認証のタイプ
 - **amqp1authNone:** 設定なし。
 - **amqp1authSASLAnonymous:** 匿名認証
 - **amqp1authSASLPlain:** ユーザー名/パスワード認証。この認証タイプでは以下のプロパティを入力する必要があります:
 - Username
 - Password
 - **amqp1authSASLExternal:** 外部認証

接続

接続は、クライアント（通常はメッセージングアプリケーションまたはサービス）がサーバー（メッセージブローカー）への TCP 接続を開始することから始まります。クライアントはサーバーのポートに接続します。通常、非 TLS 接続は 5672、TLS 接続は 5671 です。TCP 接続が確立されると、クライアントとサーバーは使用する AMQP プロトコルバージョンをネゴシエートします。AMQP 1.0.0 はさまざまなバージョンをサポートしており、ネゴシエーション中に双方がバージョン 1.0.0 の使用に合意します。

プロトコルネゴシエーションの後、サーバーの設定に応じて、クライアントは自身をサーバーに認証する必要があります場合があります。認証メカニズムには、PLAIN、EXTERNALなどのSASL(Simple Authentication and Security Layer)メカニズム、またはサーバーがサポートするその他のメカニズムが含まれます。

使用例： セキュアポート 5671 でリッスンし SASL 資格情報を使用する AMQP サーバーに接続します。

```
// 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;
```

セッション

認証されると、クライアントはAMQPセッションを開きます。セッションは、クライアントとサーバー間の通信のための論理的なコンテキストです。セッションは、関連するメッセージング操作をグループ化するために使用されます。新しいセッションを作成するには、**CreateSession** メソッドを使用します。このメソッドではセッション名を設定できます。空のままにすると、コンポーネントが自動的に1つ割り当てます。

セッションが正常に作成された場合、セッションの詳細とともに **OnAMQPSessionOpen** イベントが発生します。

```
oAMQP->CreateSession("MySession");  
oAMQP->OnAMQPSessionOpen = AMQP1AMQPSessionOpen;  
void __fastcall TMyClass::AMQP1AMQPSessionOpen(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Fr  
{  
    ShowMessage("#session-open: " + aSession->Id);  
}
```

リンク

セッション内で、クライアントはキュー、トピック、またはサーバーが提供するその他のリソースなどの特定のエンティティと通信するためのリンクを作成します。リンクは、メッセージの送受信に使用される双方向の通信チャンネルです。

コンポーネントは、送信者ノードと受信者ノードとして動作できます。各セッションに対して、**MaxLinksPerSession** プロパティで設定された上限まで、任意の数のリンクを作成できます。

セNDERリンク

新しいセNDERリンクを作成するには、**CreateSenderLink** メソッドを使用してセッション名とオプションでセNDERリンク名を渡します。リンクが正常に作成されると、**OnAMQPLinkOpen** イベントが発生します。

```
oAMQP->CreateSenderLink("MySession", "MySenderLink");  
oAMQP->OnAMQPLinkOpen = AMQP1AMQPLinkOpen;  
void __fastcall TMyForm::AMQP1AMQPLinkOpen(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *  
{  
    ShowMessage("#link-open: " + aLink->Name);  
}
```

レシーバーリンク

新しい受信者リンクを作成するには、**CreateReceiverLink** メソッドを使用し、セッションの名前と、オプションで受信者リンクの名前を渡します。リンクが正常に作成されると、**OnAMQPLinkOpen** イベントが発生します。

```
oAMQP->CreateSenderLink("MySession", "MySenderLink");
oAMQP->OnAMQPLinkOpen = AMQP1AMQPLinkOpen;
void __fastcall TMyForm::AMQP1AMQPLinkOpen(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *
{
    ShowMessage("#link-open: " + aLink->Name);
}
```

メッセージの送信

セッションが確立されリンクが作成されると、クライアントはメッセージを宛先に送信するなどのメッセージ操作を開始できます。セNDERリンクを使用してメッセージを送信するには **SendMessage** メソッドを使用します。

```
oAMQP->SendMessage("MySession", "MySenderLink", "My first AMQP Message");
```

メッセージの受信

デフォルトでは、レシーバーリンクは**自動モード**で作成されます。これは、新しいメッセージが届くたびにクライアントに配信されることを意味します。

レシーバーリンクが**手動モード**で作成されている場合は、Sync メソッド **GetMessage** を使用して新しいメッセージが届くまで取得して待機します。

AutomaticモードおよびManualモードでは、新しいメッセージが到着するたびに、イベント**OnAMQPMessage**が発生します。

```
void __fastcall OnAMQPMessageEvent(TObject *Sender, TsgcAMQP1Session *const aSession,
    TsgcAMQP1ReceiverLink *const aLink, TsgcAMQP1Message *const aMessage,
    TsgcAMQP1MessageDeliveryState &DeliveryState)
{
    ShowMessage(aMessage->ApplicationData->AMQPValue->Value);
}
```

リファレンス

- プロパティ — コンポーネントで利用可能な 5 つのプロパティ。
- メソッド — 12個のパブリックメソッド。
- イベント — 14 個の処理可能なイベント。

接続 | クライアント **AMQP1** 接続

AMQP サーバーに接続するには、まず [TsgcWebSocketClient](#) と `TsgcWSPClient_AMQP1` を作成する必要があります。その後、AMQP1 コンポーネントを WebSocket クライアントにアタッチする必要があります。

接続が成功した後、イベント `OnAMQPConnect` が発生します。

基本的な使い方

認証なしで AMQP 1.0.0 サーバーに接続します。AMQPOptions プロパティの値、仮想ホストを定義し、TsgcWebSocketClient のサーバーの Host と Port を設定します。TCP プレーン接続を使用している場合は、TsgcWebSocketClient プロパティ Specifications.RFC6455 を false に設定してください。

```
// 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;
```

認証

サーバーが認証を要求する場合は、AMQP:Authentication プロパティを使用してユーザー名/パスワードの値を設定し、AuthType を "amqp1authSASLPlain" に設定します。

```
// 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;
```

接続 | クライアント AMQP1 切断

クライアントは、以下のメソッドを使用して、現在のアクティブな接続を切断できます:

クローズ理由の送信

AMQP クライアントはサーバーに接続を閉じることを通知し、接続を閉じる理由に関する情報を提供できます。サーバーへの接続クローズをリクエストするには `Close` メソッドを使用します。

```
oAMQP.Close('invalid-frame', "The received frame has an invalid format.");
```

クローズを待機

デフォルトでは、**Close** メソッドは**非同期**です。メソッドを呼び出した後もコードは続行されます。**Close** メソッドが完了し、サーバーから確認が受信されるまで待機する場合は、Options パラメータの **Await** を `True` に設定してください。

```
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;
    }
}
```

ソケット接続を閉じます

[TsgcWebSocketClient](#) の `Active` プロパティを `False` に設定するだけです。[接続を閉じる](#) 方法について詳しく読むことができます。

接続 | アイドルタイムアウト

接続はアイドルタイムアウトのしきい値の対象となります。タイムアウトは、しきい値を超えた後、サーバーからフレームが受信されないときにクライアントによってトリガーされます。アイドルタイムアウトはミリ秒単位で測定され、最後のフレームが受信された時点から始まります。しきい値を超えると、コンポーネントはサーバーにCloseフレームを送信します。サーバーが10秒後に応答しない場合、クライアントはTCPソケットを閉じます。

アイドルタイムアウトの値は、以下のプロパティで設定できます。

AMQPOptions.IdleTimeout

このプロパティに設定された値は AMQP 接続を開くときにサーバーに送信されます。値が 0 より大きく `MaxInt` の半分より小さい場合、アイドルタイムアウトが超過していないか確認するための内部タイマーが有効になります。

例: `IdleTimeout` の値を60秒に設定します。

`AMQPOptions.IdleTimeout = 60000`

接続 | 接続状態

AMQP 1.0.0 は以下の接続状態を定義しています。

- **amqp1csUnknown:** 初期状態。
- **amqp1csStart:** この状態では接続は存在しますが、何も送受信されていません。これは、ソケット接続またはソケット受け入れを実行した直後の実装が入る状態です。
- **amqp1csHeaderReceived:** この状態では、接続ヘッダーがピアから受信されていますが、接続ヘッダーはまだ送信されていません。
- **amqp1csHeaderSent:** この状態では、接続ヘッダーがピアに送信されましたが、接続ヘッダーはまだ受信されていません。
- **amqp1csHeaderExchanged:** この状態では、接続ヘッダーがピアに送信され、ピアから接続ヘッダーが受信されています。
- **amqp1csOpenPipe:** この状態では接続ヘッダーとオープンフレームの両方が送信されましたが、何も受信されていません。
- **amqp1csOpenClosePipe:** この状態では、接続ヘッダー、open フレーム、パイプライン化された接続トラフィック、および close フレームが送信されましたが、何も受信されていません。
- **amqp1csOpenReceived:** この状態では、接続ヘッダーが交換済みです。ピアから Open フレームを受信しましたが Open フレームはまだ送信されていません。
- **amqp1csOpenSent:** この状態では接続ヘッダーが交換されています。Open フレームがピアに送信されましたが、まだ Open フレームを受信していません。
- **amqp1csClosePipe:** この状態では、接続ヘッダーが交換されています。オープンフレーム、パイプライン化された接続トラフィック、クローズフレームが送信されましたが、ピアからのオープンフレームはまだ受信されていません。
- **amqp1csOpened:** この状態では、接続ヘッダーと open フレームの両方が送受信されています。
- **amqp1csCloseReceived:** この状態では、ピアが AMQP クローズを開始したことを示す close フレームが受信されています。
接続上でこれ以上フレームが到着することは予期されませんが、フレームの送信は引き続き可能です。必要に応じて、実装はこの時点で TCP ハーフクローズを行い、接続の読み取り側をシャットダウンしてもかまいません (MAY)。
- **amqp1csCloseSent:** この状態では、クローズフレームがピアに送信されています。接続に何も書き込むことは許可されていませんが、まだ受信フレームが存在する可能性があります。必要に応じて、実装はこの時点で TCP ハーフクローズを行って接続の書き込み側をシャットダウンすることができます (MAY)。
- **amqp1csDiscarding:** DISCARDING 状態は CLOSE SENT 状態の変形で、クローズがエラーによってトリガーされます。
この場合、接続上のすべての受信フレームはピアのクローズフレームが受信されるまでサイレントに破棄されなければなりません。
- **amqp1csEnd:** この状態では、どちらのエンドポイントも接続にこれ以上何も書き込むことは許可されていません。
接続は安全に閉じて破棄できます。

AMQP クライアントには **ConnectionState** プロパティがあり、クライアントコンポーネントがどの接続状態にあるかを確認できます。

接続 | AMQP1 認証

コンポーネントには以下の認証方法があります。

- **amqp1authNone:** サーバーへの接続時に使用する認証方式がありません。
- **amqp1authSASLAnonymous:** 匿名として接続します。
- **amqp1authSASLPlain:** デフォルト。ユーザー名/パスワード認証を使用します。
- **amqp1authSASLExternal:** 現在はサポートされていません。

SASL 認証

最も一般的な認証は**amqp1authSASLPlain**タイプの使用です。この認証タイプはAMQP1コンポーネントのAMQPOptions.Authenticationプロパティからアクセスして有効にできます。

- **AuthType:** amqp1authSASLPlain を選択します。
- **Username:** SASL 認証に使用するユーザー。
- **パスワード:** SASL認証に使用するシークレット値。

SASL認証の結果は、イベント**OnAMQP1SASLAuthentication**で取得できます。

```
void __fastcall OnAMQP1SASLAuthentication(System::TObject* Sender, TsgcAMQP1SaslCode aCode,
    const System::UnicodeString aDescription, bool &Handled)
{
    ShowMessage("#sas1-authentication: " + aDescription);
}
```

接続 | Azure MessageBus

AMQP (Advanced Message Queuing Protocol) は、さまざまなノードにまたがる多様なコンテナ間の通信を促進するために設計された堅牢なメッセージングシステムです。メッセージ送信プロトコルとメッセージ自体の構造的なフレームワークの両方を標準化し、一貫性のある信頼性の高い通信を保証します。AMQP の基礎についての詳細は、AMQP 入門ガイドをご覧ください。

eSeGeCe ライブラリの AMQP コンポーネントにより、Amazon MQ や Azure Service Bus などの主要なクラウドメッセージングブローカーとのシームレスな統合が可能です。このガイドでは AMQP コンポーネントを使用して Azure Service Bus に接続し、効率的にメッセージを送受信できるマルチテナントアプリケーションを構築する方法を示します。

このコンポーネントはキュー、トピック、サブスクリプションなどの主要機能をサポートする包括的な実装を提供し、最新のIoTおよびエンタープライズアプリケーションに最適な選択肢です。

Azure Configuration

まず、Azure ポータル内でサービスバスリソースを作成します。リソースが確立されたら、統合と設定に必要なため、リソースのドメイン名をメモしておいてください。

名前空間が正常に作成された後は、Azure Portal の名前空間の概要から直接管理および監視できます。この一元化されたインターフェイスは、主要な管理ツールと設定へのアクセスを提供し、Service Bus リソースのシームレスな管理を実現します。

SAS 認証を使用する場合、ユーザー名は SAS ポリシー名で、パスワードはプライマリまたはセカンダリキーです。

```
// Create TCP client
TsgcWebSocketClient* oClient = new TsgcWebSocketClient(nullptr);
oClient->Specifications->RFC6455 = false;
oClient->Host = "esegece.servicebus.windows.net";
oClient->Port = 5671;
oClient->TLS = true;
// Create AMQP1 protocol client
TsgcWSClient_AMQP1* oAMQP1 = new TsgcWSClient_AMQP1(nullptr);
oAMQP1->Specifications->RFC6455 = false;
oAMQP1->AMQPOptions->Authentication->AuthType = amqp1authSASLPlain;
oAMQP1->AMQPOptions->Authentication->Username = "RootManageSharedAccessKey";
oAMQP1->AMQPOptions->Authentication->Password = "BhJ78+w8kMXhS/eE/nBy0cRzodx9tipbi+ASbAXIah8=";
oAMQP1->Client = oClient;
// Connect to the server
oClient->Active = true;
```

Azure CBS 認証

Azure Service Bus は、最初のSASLハンドシェイクの後で送信者と受信者を認可するために、AMQP上でClaims-Based Security (CBS) を実装しています。クライアントは\$**cbs** ノードへの管理リンクを開き、Shared Access Signature (SAS) トークンまたはMicrosoft Entra IDによって発行されたJSON Web Token (JWT) のいずれかを含む **put-token** 要求を送信します。ブローカーがトークンを検証すると、その認可はライフタイムの間キャッシュされ、アプリケーションはキュー、トピック、またはサブスクリプションに対して送信者リンクと受信者リンクの作成を進めます。

AMQP1 クライアントは2つのヘルパーメソッドを通じてこのフローを自動化します。

- **CreateAzureCbsSasToken** は CBS の送受信リンクペアを確立し、ターゲットエンティティの SAS トークンを生成して \$**cbs** に発行します。共有アクセスポリシーで認証する際に使用します。
- **CreateAzureCbsJWT** は同じCBSエクスチェンジに従いますが、JWTを \$**cbs** に送信する前に、クライアント資格情報グラントを使用してMicrosoft Entra ID (Azure AD) からアクセストークンを取得します。

両方のメソッドはアクティブな AMQP 接続が必要で、以下のパラメータを受け付けます:

- **aName:** 内部で作成される CBS リンクペアの識別子。
- **aNameSpace** と **aEntityName**: トークンオーディエンスの構築に使用される Service Bus 名前空間 (`.servicebus.windows.net` サフィックスなし) とキュー、トピック、またはサブスクリプションパス。
- **aKeyName** / **aKeyValue**: SASトークン用の共有アクセスポリシー名とキー。コンポーネントはトークンに署名し、トークンタイプ `servicebus.windows.net:sastoken` を使用して送信します。
- **aTenant**, **aApplicationId**, **aSecret**: クライアント資格情報フローでJWTをリクエストするために使用される Microsoft Entra (Azure AD) ディレクトリID、アプリケーション (クライアント) ID、およびクライアントシークレット。
- **aListeningPort** (JWT): OAuth 2.0 リダイレクト用のローカル HTTP ポート (指定しない場合はデフォルトで 8080)。
- **aExpiration** と **aTimeout**: 発行されたトークンの有効期間 (秒単位) と CBS ネゴシエーションの最大待機時間 (ミリ秒単位)。
- **aRaiseIfError**: `True` に設定されている場合、トークンの取得または CBS レスポンスが失敗すると例外を発生させます。

以下の例では、メッセージを送信する前に CBS で認証する方法を示します。

```
// 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);
```

次の例では、JWTを使用したMicrosoft Entra ID (Azure AD) 認証のみに焦点を当てます。client credentialsフローでトークンを要求し、メッセージを送受信するためのリンクを作成する前に\$**cbs**へ発行する方法を示します。

```
// 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);
```

コマンド | AMQP1 セッション

AMQP (Advanced Message Queuing Protocol) 1.0.0 仕様のコンテキストでは、セッションはクライアントとメッセージブローカー間の通信の論理コンテキストを表します。AMQP 1.0.0 セッションの内容は次のとおりです：

- **論理コンテキスト:** セッションは AMQP クライアント（プロデューサーまたはコンシューマー）と AMQP ブローカー間のメッセージング操作のための論理コンテキストを確立します。単一の接続内で関連するメッセージング操作をグループ化する方法を提供します。
- **Communication Channel:** セッションは、メッセージが送受信される通信チャンネルとして機能します。メッセージの交換、確認応答、フロー制御メカニズムをカプセル化します。
- **トランザクション境界:** セッションはメッセージ操作のトランザクション境界を定義します。複数のメッセージ送受信を単一のアトミック単位としてグループ化でき、セッション内のすべての操作が正常に処理されるか、まったく処理されないかを保証します。
- **Flow Control:** セッションは、クライアントとブローカー間でメッセージが交換されるレートを調整するフロー制御メカニズムをサポートします。フロー制御は、いずれかの当事者のリソースが圧倒されるのを防ぎ、効率的で信頼性の高いメッセージ配信を保証するのに役立ちます。
- **ライフタイム管理:** セッションには、作成時に始まり、クローズ時に終わるライフサイクルがあります。クライアントは単一の接続内で複数のセッションを確立して、メッセージ処理を並列化したり、メッセージストリームを分離したりできます。
- **リソース割り当て:** セッションはブローカー内のキュー、トピック、またはサブスクリプションなどの特定のリソースと関連付けられる場合があります。セッション内で送受信されるメッセージはこれらのリソースにバインドされ、ターゲットを絞ったメッセージルーティングと配信を可能にします。

要約すると、AMQP 1.0.0 セッションは AMQP クライアントとブローカー間のメッセージ交換のための論理的なコンテキストを提供し、メッセージングシステム内でのトランザクション整合性、フロー制御、リソース管理を促進します。メッセージング操作が実行される境界を定義し、メッセージの効率的で信頼性の高い交換を保証するのに役立ちます。

Open Session

CreateSession メソッドは、指定された名前で新しいセッションを作成します（空の場合はランダムな名前で作成します）。セッションがすでに存在する場合は例外が発生します。クライアントを使用すると、同じ AMQP 接続を使用して複数のセッションを作成できます。

セッションが正常に作成されると、**OnAMQPSessionOpen** イベントが発火します。

```
oAMQP->CreateSession("MySession");
oAMQP->OnAMQPSessionOpen = OnAMQPSessionOpenEvent;
void __fastcall TMyClass::OnAMQPSessionOpenEvent(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Frame *aFrame)
{
    ShowMessage("#session-open: " + aSession->Id);
}
```

CreateSession メソッドはセッション情報を含む **TsgcAMQP1Session** クラスを返します。

オープンセッションを待機する

デフォルトでは、**CreateSession** メソッドは非同期であるため、メソッドを呼び出した後、コードは続行されます。**CreateSession** メソッドが完了するまで待機し、サーバーから送信される確認を受信したい場合は、Options パラメータの **Await** プロパティを True に設定します。

```
void OpenSession(const System::UnicodeString aSession)
{
    TsgcAMQP1MethodOptions_SessionOpen *oOptions = new TsgcAMQP1MethodOptions_SessionOpen();
```

```

    try
    {
        oOptions->Await = true;
        AMQP1->CreateSession(aSession, oOptions);
    }
    __finally
    {
        delete oOptions;
    }
}

```

セッションを閉じる

既存のセッションを閉じるには、閉じるセッションの名前を渡して**CloseSession** メソッドを使用します。

セッションが正常に閉じられると、**OnAMQPSessionClose** イベントが発生します。

```

oAMQP->CloseSession("MySession");
oAMQP->OnAMQPSessionClose = OnAMQPSessionCloseEvent;
// OnAMQPSessionCloseEvent implementation
void __fastcall TMyClass::OnAMQPSessionCloseEvent(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Fr
{
    ShowMessage("#session-close: " + aSession->Id + " [" + IntToStr(aSession->Channel) + "] reason: " + aEnd->Err
}

```

セッションのクローズを待機

デフォルトでは、**CloseSession** メソッドは**非同期**であるため、メソッドを呼び出した後、コードは続行されます。**CloseSession**メソッドが完了するまで待機し、サーバーから送信される確認を受信したい場合は、Optionsパラメータの**Await** プロパティをTrueに設定します。

```

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 仕様では、リンクは AMQP クライアントとメッセージブローカー間の単方向通信チャンネルを表します。AMQP 1.0.0 リンクの詳細について説明します。

- **通信チャンネル:** リンクは AMQP の送信者と受信者の間でメッセージが流れる経路として機能します。送信者から受信者、またはその逆の一方方向でメッセージを送信できます。
- **単方向フロー:** 各リンクは単方向であり、メッセージはリンクに沿って一方方向にのみ移動できます。双方向通信が必要な場合は、2 つのリンクを確立する必要があります (各方向に 1 つ)。
- **メッセージ転送:** メッセージは AMQP プロトコルルールに従ってリンク間で転送されます。これらのメッセージにはペイロード、メッセージプロパティ、および通信に必要な追加のメタデータが含まれる場合があります。
- **Resource Binding:** リンクは、キュー、トピック、exchange など、AMQP ブローカー内の特定のリソースに関連付けられます。リンクを介して送受信されるメッセージは、これらのリソースに向けられるか、これらのリソースから発生します。
- **フロー制御:** リンクはメッセージの送受信レートを調整するフロー制御メカニズムをサポートしています。フロー制御により、送信者も受信者も交換されるメッセージの量に圧倒されないことが保証されます。
- **ライフタイム管理:** リンクは確立されたときに始まり、閉じられたときに終わるライフサイクルを持ちます。必要に応じて動的に作成でき、不要になったときに閉じられます。
- **Addressing:** Link は、通信のソースとターゲットのエンドポイントを指定する一意のアドレスによって識別されます。これらのアドレスにより、クライアントとブローカーは適切なエンドポイントを識別し、接続を確立できます。
- **トランザクション境界:** リンクはメッセージ操作のトランザクション境界を定義します。複数のメッセージ送受信を単一のアトミックな単位にまとめることができ、メッセージ配信の一貫性と信頼性を確保します。

まとめると、AMQP 1.0.0 リンクは AMQP クライアントとブローカー間の単方向通信手段を提供し、フロー制御、リソースバインディング、アドレス指定、メッセージングシステム内のトランザクション整合性をサポートしながらメッセージの転送を促進します。これらは AMQP プロトコルにおけるメッセージ交換の基本的な構成要素を形成します。

Linkには2種類あります:

- **セNDERリンク:** これらのリンクはメッセージの送信に使用されます。
- **レシーバーリンク:** これらのリンクはメッセージの受信に使用されます。

新しいリンクが作成または削除されるたびに、以下のイベントが発生します:

- **OnAMQPLinkOpen:** このイベントは、新しいリンクが作成されたときにトリガーされます。aLink.Mode プロパティを使用して、リンクがreceiverモードかsenderモードかを確認します。
- **OnAMQPLinkClose:** このイベントはリンクが閉じられたときにトリガーされます。

コマンド | AMQP1 センダーリンク

AMQP 1.0.0 プロトコルでは、送信側リンクはメッセージを送信する目的で AMQP クライアントと AMQP サーバーの間に確立された通信チャンネルです。クライアントとサーバー間の通信の論理チャンネルを表す AMQP セッションのコンテキスト内で動作します。

送信リンクの作成

新しいSender Linkを作成するには、次のパラメータを含む**CreateSenderLink** メソッドを呼び出します。

- **Session:** 送信リンクがアタッチされるセッション名。
- **Name:** (オプション) 送信者リンクの名前。設定されていない場合、ランダムな名前が自動的に割り当てられます。
- **Target:** (オプション) "target" パラメータを設定することでリモートホストでメッセージを受信する宛先を指定できます。ただし、特定のシナリオではターゲットの指定が不要な場合があります。そのような場合は空の文字列を指定すれば十分です。
- **SndSettleMode:** (デフォルトは mixed) AMQP はメッセージ決済メカニズムを通じて配信保証を協議する機能を提供します。リンクの確立時に、送信者と受信者の両方が決済モードを協議して合意します（各ロールに 1 つ）。送信者は以下のモードのいずれかで動作します：
 - **amqp1ssmSettled:** メッセージは送信されると正常に配信され確認されたとみなされます。
 - **amqp1ssmUnsettled:** メッセージは受信者によって明示的に受け入れまたは拒否されるまで決済されたとはみなされません。これによりメッセージの処理と取り扱いをより細かく制御できます。
 - **amqp1ssmMixed:** 単一の AMQP セッション内で決済済みモードと未決済モードの組み合わせを使用できます。メッセージが決済済みかどうかを設定するには、**SendMessage** メソッドの **MessageOptions** パラメーターを使用してください。

Sender Linkが正常に作成されると、**OnAMQPLinkOpen** イベントが発生します。

```
oAMQP1->CreateSenderLink("MySession", "MySenderLink");
oAMQP1->OnAMQPLinkOpen = AMQP1AMQPLinkOpen;
void __fastcall TMyForm::AMQP1AMQPLinkOpen(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *const aLink)
{
    ShowMessage("#link-open: " + aLink->Name);
}
```

セnderリンクの作成を待機する

デフォルトでは、**CreateSenderLink** メソッドは**非同期**です。メソッドを呼び出した後、コードは継続します。**CreateSenderLink** メソッドが完了し、サーバーから送信された確認が受信されるまで**待機**したい場合は、Options パラメーターの **Await** を **True** に設定してください。

```
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;
    }
}
```

メッセージの送信

新しいメッセージを送信するには、次のパラメータを含むSendMessageメソッドを呼び出します:

- **Session:** セッションの名前。
- **Link:** 送信側リンクの名前。
- **Text:** 文字列メッセージのテキスト。

```
oAMQP1->SendMessage("MySession", "MySenderLink", "My first AMQP Message");
```

混合モードでのメッセージ送信

Sender LinkがMixedモード（デフォルト）で作成されている場合、メッセージを送信する際に、ユーザーはメッセージを**settled**にするかしないかを設定できます。**MessageOptions** パラメータを使用して、メッセージをsettledにするかどうかを定義します。

```
TsgcAMQP1MessageOptions* oMessageOptions = new TsgcAMQP1MessageOptions();
try
{
    oMessageOptions->Settled = true;
    oAMQP1->SendMessage("MySession", "MySenderLink", "MyMessage", "message-id", oMessageOptions);
}
__finally
{
    delete oMessageOptions;
}
```

送信リンクのクローズ

既存の送信者リンクを閉じるには、以下のパラメータを持つ **CloseLink** メソッドを呼び出します:

- **Session:** リンクを含むセッションの名前。
- **Link:** 送信側リンクの名前。
- **エラー:** (オプション) リンクが閉じられた理由をここに設定できます。

セNDERリンクが正常に閉じられると、**OnAMQPLinkClose** イベントが発生します。

```
oAMQP->CloseLink("MySession", "MySenderLink");
void __fastcall TForm1::OnAMQPLinkCloseEvent(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *cc
{
    ShowMessage("#link-close: " + aLink->Name);
}
```

送信リンクのクローズを待機

デフォルトでは、**CloseLink** メソッドは**非同期**であるため、メソッドを呼び出した後、コードは続行されます。**CloseLink** メソッドが完了するまで待機し、サーバーから送信される確認を受信したい場合は、Optionsパラメータの**Await** プロパティをTrueに設定します。

```
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;  
}
```

コマンド | AMQP1 レシーバーリンク

AMQP 1.0.0 プロトコルでは、受信リンクはメッセージを受信する目的で AMQP クライアントと AMQP サーバー間に確立された通信チャンネルです。クライアントとサーバー間の通信のための論理チャンネルを表す AMQP セッションのコンテキスト内で動作します。

受信リンクの作成

新しいレシーバーリンクを作成するには、次のパラメータを含む **CreateReceiverLink** メソッドを呼び出してください:

- **Session:** 送信リンクがアタッチされるセッション名。
- **Name:** (オプション) 送信者リンクの名前。設定されていない場合、ランダムな名前が自動的に割り当てられます。
- **Source:** (オプション) ソースは、送信者として機能するリモートホスト上のノードの場所を示すように設定できます。状況によっては、このアドレスの指定が不要な場合があります。そのような場合は、パラメータの値として空の文字列を指定するだけで十分です。
- **ReadMode:** (デフォルトは `amqp1srmAuto`) レシーバーリンクはメッセージ受信のために2つのモードのいずれかで機能します:
 - **amqp1srmAuto:** 自動モード。このモードでは、レシーバーはメッセージが利用可能になったらすぐに受信するよう積極的に動作します。新しいメッセージが到着するたびに、明示的な指示なしでメッセージを自動的にリスンして受信します。
 - **amqp1srmManual:** フェッチベースモード。このモードでは、レシーバーは明示的に指示された場合にのみ新しいメッセージを取得またはフェッチします。自動モードとは異なり、レシーバーは新しいメッセージを積極的にリスンせず、次のメッセージをフェッチするための手動指示を待ちます。
- **RcvSettleMode:** (デフォルトで `amqp1srmFirst`) Receiver Linkは次のいずれかのモードで動作します:
 - **amqp1srmFirst:** メッセージが到着すると、すぐに処理され確認されます。メッセージが送信された時点でまだ確認されていなかった場合、送信者はメッセージが受信されたことを通知されます。
 - **amqp1srmSecond:** 到着したメッセージは、送信者が先に確認した後にのみ確認されます。さらに、送信時にメッセージがまだ確認されていなかった場合、送信者はメッセージが受信されたときに通知を受け取ります。

Receiver Linkが正常に作成されると、イベント **OnAMQPLinkOpen** が発生します。

```
oAMQP1->CreateReceiverLink("MySession", "MyReceiverLink");
oAMQP1->OnAMQPLinkOpen = AMQP1AMQPLinkOpen;
void __fastcall TMyForm::AMQP1AMQPLinkOpen(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *const aLink)
{
    ShowMessage("#link-open: " + aLink->Name);
}
```

受信リンクの作成を待機

デフォルトでは、**CreateReceiverLink** メソッドは非同期であるため、メソッドを呼び出した後、コードは続行されます。**CreateReceiverLink** メソッドが完了するまで待機し、サーバーから送信される確認を受信したい場合は、Optionsパラメータの**Await** プロパティをTrueに設定します。

```
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;
    }
}
```

```
}
}
```

同期メッセージ

Receiver Link が Manual ReadMode で動作している場合、**GetMessage** メソッドを呼び出して新しいメッセージを取得します。このメソッドは同期的で、タイムアウト（デフォルト 10 秒）が切れるまで待機します。メソッドが呼び出されると、コンポーネントはクレジットを 1 増やし、新しいメッセージが到着するかタイムアウトが切れるまで待機します。メッセージが到着しない場合、クレジットはゼロに戻されます。

GetMessage メソッドには以下のパラメータがあります。

- **Session:** リンクを含むセッションの名前。
- **Link :** 受信リンクの名前。
- **Timeout:** (デフォルト 1000 = 10 秒) 新しいメッセージを取得するために関数が待機する最大時間。

レシーバーリンクを閉じる

既存のレシーバーリンクを閉じるには、以下のパラメータを含む **CloseLink** メソッドを呼び出してください:

- **Session:** リンクを含むセッションの名前。
- **Link :** 受信リンクの名前。
- **エラー :** (オプション) ここでリンクが閉じられる理由を設定できます。

レシーバーリンクが正常にクローズされると、イベント **OnAMQPLinkClose** が発生します。

```
oAMQP1->CloseLink("MySession", "MyReceiverLink");
void __fastcall TForm1::OnAMQPLinkCloseEvent(TObject *Sender, TsgcAMQP1Session *const aSession, TsgcAMQP1Link *cc
{
    ShowMessage("#link-close: " + aLink->Name);
}
```

受信リンクのクローズを待機

デフォルトでは、**CloseLink** メソッドは**非同期**であるため、メソッドを呼び出した後、コードは続行されます。**CloseLink** メソッドが完了するまで待機し、サーバーから送信される確認を受信したい場合は、Options パラメータの **Await** プロパティを True に設定します。

```
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 | メッセージの送信

最初に [AMQP1 送信者リンク](#) を読んで、送信者リンクの作成方法を確認してください。

Send Message

AMQP1サーバーにテキストメッセージを送信するには、SessionとSenderLink名を渡してメソッド**SendMessage**を使用します。このメソッドには次のパラメータがあります：

- **Session:** セッションの名前。
- **Link:** 送信側リンクの名前。
- **Text:** メッセージのテキスト。
- **MessageId:** (オプション) メッセージのID。未決済モードを使用する際に、サーバーがメッセージを処理したかどうかを確認するために使用できます。
- **Options:** (オプション) メッセージ送信時のいくつかのオプションをカスタマイズできます。
 - **Settled:** 混合モードの送信リンクを使用する場合、メッセージ送信時に Settled プロパティをカスタマイズできます。
 - **Await:** メッセージが未決済で値が true の場合、サーバーによってメッセージが処理されるかタイムアウトが超過するまでコードが待機します。
 - **Timeout:** await が true の場合のミリ秒単位の値 (デフォルト 10000)。
 - **RaiseTimeoutException:** タイムアウトが超過した場合、例外が発生します (デフォルト true)。

```
oAMQP1->SendMessage("MySession", "MySenderLink", "My first AMQP Message");
```

送信メッセージを待機します

デフォルトでは、**SendMessage** メソッドはメッセージを未確認で送信する場合に非同期です。**Await** プロパティを true に設定すると、クライアントはサーバーからメッセージが処理されたことの確認を受け取るまで待機します。

```
void SendMessageAwait(const System::UnicodeString aSession, const System::UnicodeString aSenderLink, const System::UnicodeString aText)
{
    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;
    }
}
```

イベント

メッセージ送信時に、メッセージが送信されたことと、サーバーによって処理されたことを知るために使用できる2つのイベントがあります (未決済で送信する場合)。

- **OnAMQPMessageSent:** このイベントはメッセージがサーバーに送信された後に呼び出されます。SendMessage メソッドを呼び出すとメッセージは内部キューに保存され、セカンダリスレッドによって処理されます。メッセージが送信されると、このイベントが呼び出されます。
- **OnAMQPMessageSentAck:** クライアントがメッセージが AMQP1 サーバーによって処理されたという確認を受け取ったときにこのイベントが呼び出されます。

```
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 | メッセージ読み取り

新しいメッセージが受信されるたびに、**OnAMQPMessage** イベントが発火します。

TsgcAMQP1Message インスタンスには受信したメッセージが含まれます。プロパティ **aMessage.ApplicationData.AMQPValue.Value** を使用してテキストメッセージにアクセスできます。

Delivery Outcomeを指定するには、**DeliveryState**パラメータを使用します。デフォルトでは、すべてのメッセージはaccepted状態ですが、次のいずれかを設定できます：

- **amqp1mdtsAccepted**：メッセージは正常に処理されました。
- **amqp1mdtsRejected**：メッセージの処理に失敗しました。DeliveryState.Rejected プロパティを使用してエラーを設定してください。
- **amqp1mdtsReleased**：メッセージは処理されておらず、今後も処理されません。
- **amqp1mdtsModified**：amqp1mdtsReleasedと同じですが、プロパティDeliveryState.Modifiedを使用して追加データを加えることができます。

```
void __fastcall OnAMQPMessage(System::TObject* Sender, TsgcAMQP1Session* const aSession,
    TsgcAMQP1ReceiverLink* const aLink, TsgcAMQP1Message* const aMessage,
    TsgcAMQP1MessageDeliveryState &DeliveryState)
{
    if (aMessage->ApplicationData->AMQPValue->Value == "xxx")
    {
        DeliveryState.State = amqp1mdtsRejected;
        DeliveryState.Rejected->Error->Condition = "amqp-error-processing";
        DeliveryState.Rejected->Error->Description = "Value received was not the expected.";
    }
    else
    {
        DeliveryState.State = amqp1mdtsAccepted;
    }
}
```

Protocol STOMP

STOMPは Simple（または Streaming）Text Orientated Messaging Protocol の略です。STOMPは相互運用可能なワイヤーフォーマットを提供するため、STOMPクライアントは 任意のSTOMPメッセージブローカーと通信でき、多くの言語、プラットフォーム、ブローカーにわたって 容易かつ広範なメッセージング相互運用性を実現します。

当社の STOMP クライアントコンポーネントは以下の STOMP バージョンをサポートしています：1.0、1.1、1.2。

コンポーネント

TsgcWSPClient_STOMP: 汎用STOMP Protocolクライアント。任意のSTOMP Serverに接続できます。

TsgcWSPClient_STOMP_RabbitMQ: RabbitMQ ブローカー用 STOMP クライアント。

TsgcWSPClient_STOMP_ActiveMQ: ActiveMQ Broker用のSTOMPクライアント。

TsgcWSPClient_STOMP

これはクライアントプロトコルSTOMPコンポーネントです。

はじめに

これはクライアントプロトコルSTOMPコンポーネントです。このコンポーネントをフォームにドロップし、Clientプロパティを使用して [TsgcWebSocketClient](#) コンポーネントを選択する必要があります。

リファレンス

- プロパティ — コンポーネントで使用可能な8つのプロパティ。
- メソッド — 公開メソッド 11 件。
- イベント — 処理できる 6 つのイベント。

TsgcWSPClient_STOMP_RabbitMQ

これは RabbitMQ ブローカー用のクライアントプロトコル STOMP コンポーネントです。

はじめに

これは RabbitMQ ブローカー用クライアント STOMP プロトコルコンポーネントです。このコンポーネントをフォームにドロップし、Client プロパティを使用して [TsgcWebSocketClient](#) コンポーネントを選択する必要があります。

宛先

STOMP 仕様はブローカーがサポートする必要があるデスティネーションの種類を規定していません。SEND および MESSAGE フレームのデスティネーションヘッダーの値はブローカー固有です。RabbitMQ STOMP アダプターはさまざまなデスティネーションタイプをサポートしています：

- **Topic:** 一時的および永続的なトピックに対して SEND および SUBSCRIBE を行います。
- **Queue :** STOMP ゲートウェイが管理するキューへの SEND と SUBSCRIBE。
- **QueueOutside:** STOMP ゲートウェイの外部で作成されたキューへの SEND および SUBSCRIBE。
- **TemporaryQueue:** 一時的なキューを作成します (reply-to ヘッダーのみ)。
- **Exchange:** 任意のルーティングキーへのSENDと、任意のバインディング パターンへのSUBSCRIBE。

リファレンス

- プロパティ — コンポーネントで利用可能な8つのプロパティ。
- メソッド — 25個のパブリックメソッド。
- イベント — 処理できる 6 つのイベント。

TsgcWSPClient_STOMP_ActiveMQ

これは ActiveMQ ブローカー用のクライアントプロトコル STOMP コンポーネントです。

はじめに

これは ActiveMQ ブローカー用のクライアントプロトコル STOMP コンポーネントです。このコンポーネントをフォームにドロップし、Client プロパティを使用して [TsgcWebSocketClient](#) コンポーネントを選択する必要があります。

宛先

STOMP 仕様はブローカーがサポートしなければならない宛先の種類を規定していません。SEND および MESSAGE フレームの宛先ヘッダーの値はブローカー固有です。Active STOMP アダプターは複数の異なる宛先タイプをサポートしています。

- **Topic:** 一時的および永続的なトピックに対して SEND および SUBSCRIBE を行います。
- **Queue:** STOMP ゲートウェイが管理するキューへの SEND と SUBSCRIBE。

パブリッシュオプション

STOMP はできるだけシンプルになるよう設計されています。そのため、あらゆるスクリプト言語やプラットフォームが最小限の労力で他のものとメッセージをやり取りできます。STOMP では、メッセージの送受信などの各リクエストにプラグブルなヘッダーを使用できます。ActiveMQ には STOMP プロトコルへのいくつかの拡張があり、STOMP クライアントが JMS セマンティクスをサポートできるようにします。OpenWire JMS プロデューサーは STOMP コンシューマーにメッセージを送信でき、STOMP プロデューサーは OpenWire JMS コンシューマーにメッセージを送信できます。また、STOMP 同士の構成では、より豊富な JMS メッセージ制御を使用できます。

STOMP は、SENT メッセージで次の標準 JMS プロパティをサポートします。

- **CorrelationId:** 優れたコンシューマーは送信するレスポンスにこのヘッダーを追加します。
- **Expires:** メッセージの有効期限。
- **JMSXGroupID:** メッセージグループを指定します。
- **JMSXGroupSeq:** メッセージグループのシーケンス番号を指定するオプションヘッダー。
- **Persistent:** メッセージが永続的かどうか。
- **Priority:** メッセージの優先度。
- **ReplyTo:** 返信を送信する送信先。
- **MsgType:** メッセージのタイプ。

リファレンス

- プロパティ — コンポーネントで利用可能な 9 個のプロパティ。
- メソッド — 16 個のパブリックメソッド。
- イベント — 処理できるイベント 6 件。

Protocol AppRTC

WebRTC (Web Real-Time Communication) は、プラグインなしで音声通話、ビデオチャット、P2Pファイル共有のためのブラウザ間アプリケーションを可能にするために、World Wide Web Consortium (W3C) によって策定されているAPI定義です。WebRTCのRTCはリアルタイムコミュニケーションの略で、ブラウザクライアント (ピア) 間のオーディオ/ビデオストリーミングとデータ共有を可能にする技術です。一連の標準として、WebRTCはあらゆるブラウザに、プラグインやサードパーティソフトウェアをインストールすることなく、アプリケーションデータを共有し、ピアツーピアでテレビ会議を行う能力を提供します。

WebRTC コンポーネントは JavaScript API でアクセスされます。現在開発中のものとして、オーディオまたはビデオデータストリームを表す Network Stream API と、2 人以上のユーザーがブラウザ間で通信できる PeerConnection API があります。また、リアルタイムゲーム、テキストチャット、ファイル転送などの他の種類のデータ通信を可能にする DataChannel API も開発中です。

[appr.tc](#) は Google と Mozilla が開発した WebRTC デモアプリケーションであり、両方のブラウザが WebRTC API を使用して「通話」できるようにします。

コンポーネント

[TsgcWSPServer_AppRTC](#): サーバープロトコル AppRTC VCL コンポーネント。

TsgcWSPServer_AppRTC

これはサーバープロトコル AppRTC コンポーネントです。

はじめに

これはサーバープロトコル AppRTC コンポーネントです。このコンポーネントをフォームにドロップして、Server プロパティを使用して [TsgcWebSocketServer](#) コンポーネントを選択する必要があります。

パラメータ

- **IceServers:** ここで WebRTC 接続の TURN/STUN サーバーを設定できます。
- **RoomLink:** ルームにアクセスするための URL ベース。例: <https://mydemo.com/r/>
- **WebSocketURL:** WebSocket サーバーへの URL。例: <wss://mydemo.com>

WebRTC プロトコルには STUN/TURN サーバーが必要です。デモはテスト目的でパブリック STUN/TURN サーバーを使用しています。本番システムでは専用の STUN/TURN サーバーが必要です。

登録ユーザーは **Windows 用の Coturn** サーバーのコンパイル済みバイナリをダウンロードできます。[COTURN STUN/TURN](#) の詳細をお読みください。

IceServers の設定

STUN/TURN サーバーが次の IP アドレスで動作している場合: 51.122.4.88 でポート 3478 でリッスン。接続するユーザーは「apprtc」で認証情報は「secret」です。IceServers を次のように設定します:

```
{
  "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"
}
```

リファレンス

- プロパティ — コンポーネントで利用可能な 5 つのプロパティ。
- メソッド — 0 つのパブリックメソッド。
- イベント — 処理可能な 11 個のイベント。

プロトコル WebRTC

WebRTC (Web Real-Time Communication) は、プラグインなしで音声通話、ビデオチャット、P2Pファイル共有のためのブラウザ間アプリケーションを可能にするために、World Wide Web Consortium (W3C) によって策定されているAPI定義です。WebRTCのRTCはリアルタイムコミュニケーションの略で、ブラウザクライアント (ピア) 間のオーディオ/ビデオストリーミングとデータ共有を可能にする技術です。一連の標準として、WebRTCはあらゆるブラウザに、プラグインやサードパーティソフトウェアをインストールすることなく、アプリケーションデータを共有し、ピアツーピアでテレビ会議を行う能力を提供します。

WebRTC コンポーネントは JavaScript API でアクセスされます。現在開発中のものとして、オーディオまたはビデオデータストリームを表す Network Stream API と、2 人以上のユーザーがブラウザ間で通信できる PeerConnection API があります。また、リアルタイムゲーム、テキストチャット、ファイル転送などの他の種類のデータ通信を可能にする DataChannel API も開発中です。

コンポーネント

[TsgcWSPServer_WebRTC](#): サーバープロトコル WebRTC VCL コンポーネント。

パラメータ

- **IceServers**: WebRTC 接続の TURN/STUN サーバーを設定できます。デフォルトでは以下のパブリック STUN サーバーを使用します

```
{"iceServers": [{"url": "stun:stun.l.google.com:19302"}]}
```

ブラウザテスト

このプロトコルをお好みの Web ブラウザでテストするには、この URL を入力してください (カスタムのホストとポートを定義する必要があります)。

<http://host:port/webrtc.esegece.com.html>

TsgcWSPServer_WebRTC

これはサーバープロトコル WebRTC コンポーネントです。

はじめに

これはサーバープロトコル WebRTC コンポーネントです。このコンポーネントをフォームに配置し、Server プロパティを使用して [TsgcWebSocketServer](#) コンポーネントを選択する必要があります。

WebRTC プロトコルには STUN/TURN サーバーが必要です。デモはテスト目的でパブリック STUN/TURN サーバーを使用しています。本番システムでは専用の STUN/TURN サーバーが必要です。

登録ユーザーは **Windows 用の Coturn** サーバーのコンパイル済みバイナリをダウンロードできます。[COTURN STUN/TURN](#) の詳細をお読みください。

リファレンス

- プロパティ — コンポーネントで利用可能な5つのプロパティ。
- メソッド — 2つのパブリックメソッド。
- イベント — 11個の処理可能なイベント。

プロトコル WebRTC Javascript

以下に使用可能なメソッドがあります。{%host%} と {%port%} 変数を必要に応じて置き換えてください。例: sgcWebSocket サーバーが www.example.com の port 80 でリッスンするように設定している場合は、次のように設定する必要があります。

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
<script src="http://www.example.com:80/webrtc.esegce.com.js"></script>
```

接続を開く

WebSocket 接続が開かれると、ブラウザはローカルのカメラとマイクへのアクセスを要求します。アクセスを許可する必要があります。

```
<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>
```

WebRTC チャンネルを開く

ブラウザがローカルのカメラとマイクにアクセスできる場合、'sgcmediastart' イベントが発生し、その後 webrtc_connect プロシーダを使用して別のクライアントへの接続を試みることができます

```
<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 チャンネルを閉じる

```
<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>
```

プロトコル WAMP

WAMP は、RPC と PubSub の 2 つの非同期メッセージングパターンを提供するオープンな WebSocket サブプロトコルです。

技術的には、WAMP は JSON をメッセージシリアル化フォーマットとして使用する公式に登録された WebSocket サブプロトコル (WebSocket 上で動作) です。

RPC とは何ですか？

リモートプロシージャコール (RPC) は、クライアントとサーバーの2つの役割を持つピア間のメッセージングパターンです。

サーバーは、既知のエンドポイントで呼び出すメソッドまたは手続きを提供します。

クライアントは、メソッドまたはプロシージャのエンドポイントと呼び出しの引数を指定して、リモートメソッドまたはプロシージャを呼び出します。

サーバーはコールに提供された引数を使用してメソッドまたはプロシージャを実行し、コールの結果をクライアントに返します。

PubSubとは何か

Publish & Subscribe (PubSub) は、パブリッシャー、サブスクライバー、ブローカーの 3 つの役割を持つピアが関与するメッセージングパターンです。

パブリッシャーは、特定のピアではなく抽象的なアドレスとしてのトピック (別名チャンネル) を提供することで、イベントを送信 (発行) します。

サブスクライバーは、まず関心のあるトピック (チャンネルとも呼ばれます) を提供することでイベントを受信します。その後、サブスクライバーはそのトピックに発行されたイベントを受信します。

ブローカーはパブリッシャーとサブスクライバーの間に位置し、メッセージの発行をサブスクライバーに仲介します。ブローカーはトピックごとのサブスクライバーリストを維持し、新しく発行されたイベントを適切なサブスクライバーにディスパッチします。

ブローカーが自ら、たとえばブローカーがRPCサーバーとしても機能し、サーバーで実行されたメソッドが PubSub イベントをトリガーすべき場合に、イベントをディスパッチすることもあります。

まとめると、PubSub は中間者であるブローカーを介してパブリッシャーとレシーバーを分離します。

コンポーネント

[TsgcWSPServer_WAMP](#): サーバープロトコル WAMP VCL コンポーネント。

[TsgcWSPClient_WAMP](#): クライアントプロトコル WAMP VCL コンポーネント。

[Javascript コンポーネント](#): クライアント Javascript リファレンス。

最も一般的な用途

- **RPC**
 - [シンプル RPC](#)
 - [RPC Progress Results](#)
- **PubSub**
 - [サブスクライバー](#)
 - [パブリッシャー](#)

ブラウザテスト

このプロトコルをお気に入りのWebブラウザでテストする場合は、次のURLを入力してください(カスタムのホストとポートを定義する必要があります)

`http://host:port/wamp.esegece.com.html`

TsgcWSPServer_WAMP

これはサーバープロトコルWAMPコンポーネントです。

はじめに

これはServer Protocol WAMPコンポーネントです。このコンポーネントを フォームに配置し、Serverプロパティを使用して[TsgcWebSocketServer](#) コンポーネントを選択する必要があります。

リファレンス

- プロパティ — コンポーネントで利用可能な4つのプロパティ。
- メソッド — 4個のパブリックメソッド。
- イベント — 処理可能な14のイベント。

TsgcWSPClient_WAMP

これはクライアントプロトコル WAMP コンポーネントです。

はじめに

これはクライアントプロトコルWAMPコンポーネントです。フォームにこのコンポーネントを配置し、Client プロパティを使用して [TsgcWebSocketClient](#) コンポーネントを選択する必要があります。

リファレンス

- プロパティ — コンポーネントで利用可能な3つのプロパティ。
- メソッド — 7 個のパブリックメソッド。
- イベント — 処理できる13個のイベント。

プロトコル WAMP JavaScript

以下に使用可能なメソッドがあります。{%host%} と {%port%} 変数を必要に応じて置き換えてください。例: sgcWebSocket サーバーが www.example.com の port 80 でリッスンするように設定している場合は、次のように設定する必要があります。

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
<script src="http://www.example.com:80/wamp.esegce.com.js"></script>
```

接続を開く

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegce.com.js"></script>
<script>
  var socket = new sgcws_wamp('ws://{%host%}:{%port%}');
</script>
```

新しいプレフィックスを送信

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegce.com.js"></script>
<script>
  var socket = new sgcws_wamp('ws://{%host%}:{%port%}');
  socket.prefix('sgc', 'http://www.esegce.com');
</script>
```

RPC（リモートプロシージャコール）のリクエスト

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegce.com.js"></script>
<script>
  var socket = new sgcws_wamp('ws://{%host%}:{%port%}');
  socket.call('', 'sgc:CallTest', '20')
</script>
```

TopicURI をサブスクライブする

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegce.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>
```

メッセージを発行

```
<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>
```

受信メッセージのアラートを表示

```
<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 アラートを表示

```
<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イベントのアラート表示

```
<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>
```

接続を閉じる

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
<script>
  socket.close();
</script>
```

接続ステータスを取得する

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/wamp.esegece.com.js"></script>
<script>
  socket.state();
</script>
```

WAMP | サブスクライバー

サブスクライバーは、まず関心のあるトピック（チャンネルとも呼ばれます）を提供することでイベントを受信します。その後、サブスクライバーはそのトピックに発行されたイベントを受信します。トピックからイベントを受信するには、サブスクライバーはまずそのトピックを購読する必要があります。

WAMP クライアント

```
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 サーバー

```
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 | パブリッシャー

パブリッシャーは、特定のピアではなく抽象アドレスとしてトピック（別名チャンネル）を提供することでイベントを送信（パブリッシュ）します。Publishメソッドを呼び出し、トピック名と送信したいメッセージを引数として渡すだけです。このメッセージはそのトピックのすべてのサブスクライバーに配信されます。なお、トピックにメッセージをパブリッシュするためにそのトピックにサブスクライブする必要はありません。

サーバー側では何も設定する必要はありません。公開メッセージが受信されると、メッセージは自動的にクライアントにブロードキャストされます。

WAMP クライアント

```
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コンポーネントの最も一般的な用途は、クライアントがサーバーにメソッドをリクエストし、サーバーがクライアントに応答を送信することです。クライアントはメソッドの名前のみを送信したり、サーバーが結果を計算するために必要なパラメータを渡したりできます。サーバーはリクエストを処理し、成功した場合は結果とともにクライアントに応答を送信します。エラーがある場合、サーバーはクライアントにエラー応答を送信します。

わかるように、リクエストは1つで応答も1つ(成功または失敗)です。

例：サーバーに **GetTime** というメソッドがあり、クライアントがこのメソッドをリクエストするたびにサーバーはサーバー時刻を返します。

WAMP サーバー

```
void OnServerCall(TsgcWSConnection *Connection, const string CallId, const string ProcUri, const string Arguments)
{
    if (ProcUri == "GetTime")
    {
        oServerWAMP->CallResult(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 クライアント

```
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 プログレス結果

リモートプロシージャコールが完了するために複数の結果が必要な場合があります。デフォルトでは WAMP 1.0 プロトコルは呼び出しで部分的な結果を許可しませんが、これは sgcWebSockets ライブラリの専用機能です。フローは単純な RPC と非常に似ていますが、ここでは `CallResult` が呼び出されてプロセスが終了する前に 1 つ以上の部分的な結果があります。

基本的に、クライアントはサーバーにプロシージャをリクエストし、サーバーは結果またはエラーを送信できます。結果を送信する場合、それが最終結果であるか、後でさらに結果を送信する必要があります。最終結果の場合、**`CallResult`** メソッドを呼び出してプロセスが終了します。送信する結果がさらにある場合は、**`CallProgressResult`** メソッドを呼び出します。

使用例: クライアントがサーバーに1秒ごとにサーバー時刻を受信し、20件のメッセージの後に停止するメソッドをリクエストします。

WAMP サーバー

```
void OnServerCall(TsgcWSConnection *Connection, const string CallId, const string ProcUri, const string Arguments)
{
    if (ProcUri == "GetProgressiveTime")
    {
        int vNum = StrToInt(Arguments);
        for (int i = 1; i = vNum; i++)
        {
            if (i == 20)
            {
                oServerWAMP->CallResult(CallId, FormatDateTime("yyymmdd hh:nn:ss", Now));
            }
            else
            {
                oServerWAMP->CallProgressiveResult(CallId, FormatDateTime("yyymmdd hh:nn:ss", Now));
            }
        }
    }
    else
    {
        oServer->WAMP->CallError(CallId, "Unknown method");
    }
}

oServer = new TsgcWebSocketServer();
oServer->Port = 80;
oServerWAMP = new TsgcWSPServer_WAMP();
oServerWAMP->OnCall = OnServerCallEvent();
oServerWAMP->Server = oServer;
oServer->Active = true;
```

WAMP クライアント

```
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");
```

Protocol WAMP 2

WAMP は、異なる言語で動作するオープンな WebSocket プロトコルで統一的なアプリケーションルーティングを提供します。

WAMP を使用すると、疎結合で（ソフト）リアルタイムで通信するアプリケーションコンポーネントから分散システムを構築できます。

WAMPはその中核として、アプリケーションコンポーネントが互いに通信するための2つの通信パターンを提供します。

- パブリッシュ & サブスクライブ (PubSub)
- リモートプロシージャコール (RPC)

WAMP は使いやすく、実装が簡単で、最新のウェブ標準 (WebSocket、JSON、URI) に基づいています。

コンポーネント

[TsgcWSPClient_WAMP2](#): クライアントプロトコル WAMP2 VCL コンポーネント。

TsgcWSPClient_WAMP2

これはクライアントプロトコル WAMP2 コンポーネントです。

はじめに

これはクライアントプロトコル WAMP2 コンポーネントです。このコンポーネントをフォームにドロップし、Client プロパティを使用して [TsgcWebSocketClient](#) コンポーネントを選択する必要があります。

セッションメソッド

- **ABORT:** ルーターとクライアントの両方が ABORT メッセージを送信することで WAMP セッションの開始を中止できます。

Reason は URI でなければなりません。

Details は、追加のオプションとしてクローズ情報を提供できるディクショナリである必要があります (以下を参照)。

ABORT メッセージへのレスポンスは期待されません。

- **GOODBYE:** WAMP セッションは、ルーターがクライアントに WELCOME メッセージを送信することでライフタイムを開始し、基礎となるトランスポートが消滅したとき、またはいずれかのピアが GOODBYE メッセージを送信し、もう一方のピアがそれに応答して GOODBYE メッセージを返したときに終了します。

Reason は URI でなければなりません。

Details は、追加のオプションのクローズ情報を提供できるディクショナリでなければなりません。

パブリッシュ/サブスクライブメソッド

- **PUBLISH:** パブリッシャーがあるトピックにイベントを発行するよう要求するとき、ブローカーに PUBLISH メッセージを送信します：

Request は、パブリッシャーが選択したランダムな一時 ID で、ブローカーのレスポンスとリクエストを関連付けるために使用されます。

Options は、拡張可能な方法で追加の公開リクエストの詳細を提供できるディクショナリです。詳細は以下で説明します。

Topic はパブリッシュされるトピックです。

Arguments はアプリケーションレベルのイベントペイロード要素のリストです。リストの長さはゼロになる場合があります。

ArgumentsKw は、キーワード引数として提供されるアプリケーションレベルのイベントペイロードを含むオプションの辞書です。辞書は空でも構いません。

ブローカーが発行を実行・許可できる場合、ブローカーは発行されたイベントのトピックの現在のすべてのサブスクライバーにイベントを送信します。

デフォルトでは、パブリケーションは確認応答されず、ブローカーはパブリケーションが実際に成功したかどうかを応答しません。

- **SUBSCRIBE:** サブスクライバーは、SUBSCRIBE メッセージを送信することで、トピックへの関心をブローカーに伝えます。

Request はサブスクライバーが選択したランダムなエフェメラル ID であり、ブローカーのレスポンスをリクエストと関連付けるために使用する必要があります。

Options は、拡張可能な方法で追加のサブスクリプションリクエスト詳細を提供できるディクショナリでなければなりません。

Topic はサブスクライバーがサブスクライブしたいトピックで、URI でなければなりません。

- **UNSUBSCRIBE:** 購読者がサブスクリプションのイベント受信に興味がなくなったとき、UNSUBSCRIBEメッセージを送信します

Request はサブスクライバーが選択したランダムなエフェメラル ID であり、ブローカーのレスポンスをリクエストと関連付けるために使用する必要があります。

SUBSCRIBED.Subscription は、もともとブローカーからサブスクライバーに渡されたサブスクリプションの ID でなければなりません。

RPC メソッド

- **CALL:** 呼び出し元 (Caller) がリモートプロシージャを呼び出す場合、ディーラーにCALLメッセージを送信します:

Request は、呼び出し元が選択したランダムなエフェメラル ID であり、ディーラーのレスポンスとリクエストを関連付けるために使用されます。

Options は、拡張可能な方法で呼び出しリクエストの追加詳細を提供するためのディクショナリです。詳細については後述します。

Procedure は呼び出されるプロシージャの URI です。

Arguments は位置引数のリストです (各要素は任意の型)。リストの長さはゼロでも構いません。

ArgumentsKw はキーワード呼び出し引数のディクショナリです (各引数は任意の型)。ディクショナリは空でも構いません。

- **REGISTERCALL:** Callee は REGISTER メッセージを送信することで、Dealer とのエンドポイントの可用性を宣言します。

Requestは呼び出し先 (Callee) によって選ばれたランダムな一時IDで、ディーラーのレスポンスとリクエストを関連付けるために使用されます。

Options は、拡張可能な方法で追加の登録リクエストの詳細を提供できる辞書です。詳細は後述します。

Procedure は Callee が登録するプロシージャです。

- **UNREGISTERCALL:** 呼び出し先が登録されたプロシージャの実装を提供する意思がなくなった場合、Dealer に UNREGISTER メッセージを送信します:

Requestは呼び出し先 (Callee) によって選ばれたランダムな一時IDで、ディーラーのレスポンスとリクエストを関連付けるために使用されます。

REGISTERED.Registration は、もともとディーラーから Callee に渡された、取り消す登録の ID です。

- **INVOCATION:** Dealerが呼び出しを実行 (仲介) でき、それを許可する場合、手続きを実装するそれぞれのCalleeにINVOCATIONメッセージを送信します。

Requestは、Dealerによって選択されたランダムで一時的なIDで、Calleeの応答をリクエストと関連付けるために使用されます。

REGISTERED.Registrationは、手続きがDealerに登録された際の登録IDです。

Details は、拡張可能な方法で呼び出しリクエストの追加詳細を提供するためのディクショナリです。詳細については後述します。

CALL.Arguments は呼び出し元が提供した元の位置引数のリストです。

CALL.ArgumentsKw は、呼び出し元が提供した元のキーワード呼び出し引数の辞書です。

- **YIELD:** Callee が呼び出しを正常に処理して実行を終了できた場合、Dealer に YIELD メッセージを送信して応答します。

INVOCATION.Requestは、元のinvocationリクエストのIDです。

Options は追加オプションを提供するための辞書です。

Arguments は位置的な結果要素 (任意の型) のリストです。リストの長さはゼロでも構いません。

ArgumentsKw はキーワード結果要素の辞書 (各要素は任意の型) です。辞書は空でも構いません。

リファレンス

- プロパティ — コンポーネントで利用可能な3個のプロパティ。
- メソッド — 11 個のパブリックメソッド。
- イベント — 処理できる20個のイベント。

プロトコルデフォルト

これは "JSONRPC 2.0" メッセージを使用して実装されたデフォルトのサブプロトコルです。このプロトコルを使用してメッセージを送信するたびに、以下のプロパティを持つ JSON オブジェクトが作成されます:

jsonrpc: JSON-RPC プロトコルのバージョンを指定する文字列。正確に「2.0」でなければなりません。

method: 呼び出されるメソッドの名前を含む文字列。「rpc」という語の後にピリオド文字 (U+002E または ASCII 46) が続くメソッド名は、rpc 内部メソッドと拡張機能のために予約されており、他の目的には使用してはなりません。

params: メソッドの呼び出し中に使用されるパラメーター値を保持する構造化された値。このメンバーは省略してもかまいません。

id: クライアントによって設定される識別子で、含まれる場合は文字列、数値、または NULL 値を含む必要があります。含まれない場合は通知とみなされます。値は通常 Null であるべきではなく [1]、数値には小数部分を含めないべきです [2]

JSON オブジェクトの例:

```
{"jsonrpc": "2.0", "method": "subtract", "params": [42, 23], "id": 1}
```

機能

- パブリッシュ/サブスクライブメッセージパターンで 1 対多のメッセージ配布とアプリケーションの切り離しを実現します。ワイルドカード文字をサポートしているため、チャンネルの階層にサブスクライブできます。例えば、'news' で始まるすべてのチャンネルにサブスクライブしたい場合は `Subscribe('news*')` を呼び出してください。
- ペイロードの内容に **依存しない** メッセージトランスポート
- Acknowledgment** 送信されたメッセージの。
- サーバーローカルトランザクションを通じた **トランザクションメッセージ**をサポートします。クライアントがトランザクションをコミットすると、サーバーはキューに入れられたすべてのメッセージを処理します。クライアントがトランザクションをロールバックすると、すべてのメッセージが削除されます。
- メッセージ配信のための **QoS** (Quality of Service) を実装します。

コンポーネント

[TsgcWSPClient_sgc](#): サーバープロトコルデフォルト VCL コンポーネント。

[TsgcWSPClient_sgc](#): クライアントプロトコルデフォルト VCL コンポーネント。

[Javascript コンポーネント](#): クライアント Javascript リファレンス。

ブラウザテスト

このプロトコルをお気に入りのWebブラウザでテストする場合は、次のURLを入力してください(カスタムのホストとポートを定義する必要があります)

`http://host:port/esegece.com.html`

TsgcWSPServer_sgc

これはサーバー プロトコルのデフォルト コンポーネントです。

はじめに

これはサーバープロトコルのデフォルトコンポーネントです。このコンポーネントをフォームにドロップし、Server プロパティを使用して [TsgcWebSocketServer](#) コンポーネントを選択する必要があります。

リファレンス

- プロパティ — コンポーネントで利用可能な6個のプロパティ。
- メソッド — 6個のパブリックメソッド。
- イベント — 処理可能な16個のイベント。

TsgcWSPClient_sgc

これはクライアントプロトコルデフォルトコンポーネントです。

はじめに

これはクライアントプロトコルのデフォルトコンポーネントです。このコンポーネントをフォームにドロップし、Clientプロパティを使用して[TsgcWebSocketClient](#) コンポーネントを選択する必要があります。

リファレンス

- プロパティ — コンポーネントで利用可能な 5 つのプロパティ。
- メソッド — 11 のパブリックメソッド。
- イベント — 処理できる 15 のイベント。

TsgclWWSPClient_sgc

これは Inraweb クライアントプロトコルのデフォルトコンポーネントです。

はじめに

これはInrawebクライアントプロトコルのデフォルトコンポーネントです。このコンポーネントをフォームにドロップし、 Clientプロパティを使用して [TsgclWWebSocketClient](#) コンポーネントを選択する必要があります。

リファレンス

- プロパティ — コンポーネントで利用可能な2つのプロパティ。
- メソッド — 3つのパブリックメソッド。
- イベント — 処理可能な6つのイベント。

プロトコルデフォルト JavaScript

デフォルトプロトコル Javascript sgcWebSockets は **sgcWebSocket.js** と **esegece.com.js** ファイルを使用します。

以下に使用可能なメソッドがあります。{%host%} と {%port%} 変数を必要に応じて置き換えてください。例: sgcWebSocket サーバーが www.example.com の port 80 でリッスンするように設定している場合は、次のように設定する必要があります。

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
<script src="http://www.example.com:80/esegece.com.js"></script>
```

接続を開く

```
<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>
```

Send Message

```
<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>
```

受信メッセージのアラートを表示

```
<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>
```

受信したイベントメッセージのアラートを表示

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcws('ws://{%host%}:{%port%}');
  socket.on('sgcevent', function(event)
  {
    alert('channel:' + event.channel + '. message: ' + event.message);
  }
</script>
```

RPCを呼び出す

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcws('ws://{%host%}:{%port%}');
  var params = {param:10};
  socket.rpc(GUID(), 'test', JSON.stringify(params));
</script>
```

RPC レスポンスの処理

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcws('ws://{%host%}:{%port%}');
  socket.on('sgcrpcresult', function(event)
  {
    alert('result:' + event.result);
  }
  socket.on('sgcrpcerror', function(event)
  {
    alert('error:' + event.code + ' ' + event.message);
  }
</script>
```

Call Notify

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcws('ws://{%host%}:{%port%}');
  var params = {param:10};
  socket.notify('test', JSON.stringify(params));
</script>
```

トランザクションでメッセージを送信

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
  var socket = new sgcws('ws://{%host%}:{%port%}');

  socket.starttransaction('sgc:test');
  socket.publish('Message1', 'sgc:test');
  socket.publish('Message2', 'sgc:test');
```

```
socket.publish('Message3', 'sgc:test');
socket.commit('sgc:test');
</script>
```

チャンネルへのサブスクライブまたはサブスクライブ解除時にアラートを表示します。

```
<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('sgcsubscribe', function(event)
  {
    alert('subscribed: ' + event.channel);
  }
  socket.on('sgcunsubscribe', function(event)
  {
    alert('unsubscribed: ' + event.channel);
  }
</script>
```

OnConnect、OnDisconnect、OnErrorイベントのアラート表示

```
<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('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>
```

セッションの取得

```
<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('sgcsession', function(event)
  {
    alert(event.guid);
  });
  socket.getsession();
</script>
```

接続を閉じる

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/esegece.com.js"></script>
<script>
```

```
socket.close();
</script>
```

接続ステータスを取得する

```
<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.state();
</script>
```

QoSの設定

```
<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.qoslevel1();
  socket.publish('message', 'channel');
</script>
```

キューレベルの設定

```
<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.queuelevel2();
  socket.publish('message1', 'channel1');
  socket.publish('message2', 'channel1');
</script>
```


プロトコルデータセット

このプロトコルはプロトコルデフォルトを継承し、このプロトコルに接続されたクライアント間でデータセットの変更をブロードキャストしたい場合に便利です。2つのモードで使用できます。

- 1. データベースのレプリケーション:** データベースの変更がすべてのクライアントデータベースにレプリケートされます。例えば、サーバーが株価データベースを持ち、すべての接続クライアントが株価の変更を受信します。サーバーに1つのデータベースがあり、各クライアントは独自のデータベースを持ちます。株価が更新されるたびに、この変更がすべての接続クライアントにブロードキャストされ、各クライアントが自身のレコードデータベースを更新します。このモードタイプには UpdateMode: upWhereAll または upWhereChanged を使用します。
- 2. データベースの更新:** サーバーとクライアントが共有する単一のデータベースがあり、クライアントがデータベースのレコードを更新するたびに、他のすべてのクライアントにその更新が通知されます。このモードには UpdateMode: upRefreshAll を使用してください。

最も一般的な使用方法

- **Update Mode**
 - [テーブルレプリケートの方法](#)
 - [更新通知の方法](#)

「JSON-RPC 2.0」オブジェクトを使用し、データセットの変更があるたびに、Dataset オブジェクトを使用してすべてのフィールド値（*サポートされているフィールドのみ）を送信します。

コンポーネントがデータセット上でレコードを検索できるようにするには、どのフィールドがキーであるかを指定する必要があります。例: データセットの ID フィールドがキーの場合は、次のようなコードを記述する必要があります

```
void OnAfterOpenDataSet(TDataSet *DataSet)
{
    DataSet->FieldByName("ID")->ProviderFlags =
        DataSet->FieldByName("ID")->ProviderFlags + [pfInKey];
}
```

コンポーネント

[TsgcWSPServer_Dataset](#): サーバープロトコル Dataset VCL コンポーネント。

[TsgcWSPClient_Dataset](#): クライアントプロトコル Dataset VCL コンポーネント。

[JavaScript コンポーネント](#): クライアント JavaScript リファレンス。

ブラウザテスト

このプロトコルをお気に入りのWebブラウザでテストする場合は、次のURLを入力してください(カスタムのホストとポートを定義する必要があります)

`http://host:port/dataset.esegece.com.html`

TsgcWSPServer_Dataset

これはサーバープロトコルデータセットコンポーネントです。

はじめに

これはサーバープロトコル Dataset コンポーネントです。このコンポーネントをフォームにドロップし、Server プロパティを使用して [TsgcWebSocketServer](#) コンポーネントを選択し、Dataset プロパティを使用して Dataset コンポーネントを選択する必要があります。

このコンポーネントは [TsgcWSProtocol_Server_sgc](#) からすべてのメソッドとプロパティを継承します。

リファレンス

- プロパティ — コンポーネントで利用可能なプロパティ 13 件。
- メソッド — 9 つのパブリックメソッド。
- イベント — 処理できるイベント 22 件。

TsgcWSPClient_Dataset

これはクライアントプロトコルデータセットコンポーネントです。

はじめに

これはクライアントプロトコルデータセットコンポーネントです。このコンポーネントをフォームにドロップし、Client プロパティを使用して [TsgcWebSocketClient](#) コンポーネントを選択し、Dataset プロパティを使用してデータセットコンポーネントを選択する必要があります。

このコンポーネントは [TsgcWSProtocol_Client_sgc](#) からすべてのメソッドとプロパティを継承します。

リファレンス

- プロパティ — コンポーネントで利用可能な13のプロパティ。
- メソッド — 14 個のパブリックメソッド。
- イベント — 処理可能な 25 個のイベント。

TsgclWWSPClient_Dataset

これは IntraWeb クライアントプロトコルデータセットコンポーネントです。

はじめに

これは IntraWeb クライアントプロトコル Dataset コンポーネントです。このコンポーネントをフォームにドロップし、Client プロパティを使用して [TsgclWWWebSocketClient](#) コンポーネントを選択し、Dataset プロパティを使用して Dataset コンポーネントを選択する必要があります。

このコンポーネントは [TsgclWWSPClient_sgc](#) からすべてのメソッドとプロパティを継承します。

リファレンス

- プロパティ — コンポーネントで利用可能な3つのプロパティ。
- メソッド — 6つのパブリックメソッド。
- イベント — 処理できるイベント9件。

プロトコルデータセット Javascript

Dataset プロトコル Javascript sgcWebSockets は **sgcWebSocket.js** および **dataset.esegece.com.js** ファイルを使用します。

以下に使用可能なメソッドがあります。{%host%} と {%port%} 変数を必要に応じて置き換えてください。例: sgcWebSocket サーバーが www.example.com の port 80 でリッスンするように設定している場合は、次のように設定する必要があります。

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
<script src="http://www.example.com:80/dataset.esegece.com.js"></script>
```

接続を開く

```
<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>
```

Send Message

```
<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>
```

受信メッセージのアラートを表示

```
<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>
```

受信したデータセットでアラートを表示する

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
  var socket = new sgcws_dataset('ws://{%host%}:{%port%}');
  socket.on('sgcmessage', function(event)
  {
    alert(event.message);
  })
</script>
```

```
}
</script>
```

チャンネルへのサブスクライブまたはサブスクライブ解除時にアラートを表示します。

```
<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イベントのアラート表示

```
<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>
```

全データセット変更の購読

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
  socket.subscribe_all();
</script>
```

すべての Dataset 変更のサブスクライブを解除します。

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
  socket.unsubscribe_all();
</script>
```

データセットの変更を処理する

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
var socket = new sgews_dataset('ws://{%host%}:{%port%}');

socket.on('sgcdataset', function(evt){

if ((evt.channel == "sgc@dataset@new") || (evt.channel == "sgc@dataset@update")) {

... here you need to implement your own code insert/update records ...
}
else if (evt.channel == "sgc@dataset@delete") {

... here you need to implement your own code to delete records ...

}
});
</script>
```

接続を閉じる

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
    socket.close();
</script>
```

接続ステータスを取得する

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/dataset.esegece.com.js"></script>
<script>
    socket.state();
</script>
```

プロトコル Dataset | テーブルレプリケート

このモードは、接続されているすべてのクライアントに対してテーブルが複製されるという一般的なシナリオを解決しようとするものです。例えば、株価テーブルを持つサーバーがあり、株価の変更をブロードキャストしてすべてのクライアントに送信したいが、クライアントがデータベースに接続できることは望まない場合です。そのため、いずれかの株価に変更があるたびに、レコード情報がブロードキャストされ、接続されているすべてのクライアントに送信されます。各クライアントはそのレコードを読み取り、自身のテーブルを更新します。

Demos フォルダの SQLite/MultipleDatabase デモをご確認ください。

データセットサーバーの設定

新しいDataset プロトコルサーバーを作成し、次のプロパティを使用して設定します

- **ApplyUpdates: True** に設定すると、変更があるたびにクライアントにブロードキャストされます
- **AutoSynchronize: True** に設定すると、新しいクライアントがサーバーに接続するたびに、サーバーはすべてのレコード（メタデータとデータ）を送信し、クライアントはサーバーから最新情報を取得します。
- **UpdateMode: upWhereAll** または **upWhereChanged** に設定します。違いは、前者はレコードのすべてのフィールドを送信し、後者は更新で変更されたフィールドのみを送信します。

```
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;
```

Dataset クライアントを設定する

新しいデータセットプロトコルクライアントを作成し、以下のプロパティを使用して設定します

- **ApplyUpdates: True** に設定すると、変更があるたびにサーバーに送信されます。
- **AutoSubscribe: True** に設定すると、新しいクライアントがサーバーに接続するたびに、クライアントは自動的に更新、削除、新しいレコードにサブスクライブします。
- **UpdateMode: upWhereAll** または **upWhereChanged** に設定します。違いは、前者はレコードのすべてのフィールドを送信し、後者は更新で変更されたフィールドのみを送信します。

```
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;
```


プロトコルデータセット | 更新の通知

このモードは、サーバーとクライアントが単一のデータベースを共有するシナリオ（サーバーとクライアントが同じ物理データベースに接続）で、クライアントが他のクライアントがデータセットに変更を加えるたびに通知を受けたい場合の解決策を試みます。

Demos フォルダの SQLite/SingleDatabase デモを確認できます。

データセットサーバーの設定

新しいDataset プロトコルサーバーを作成し、次のプロパティを使用して設定します

- **ApplyUpdates: True** に設定すると、変更があるたびにクライアントにブロードキャストされます
- **AutoSynchronize: False** に設定してください。ここでは True に設定する必要はありません。クライアントがサーバーと同じデータベースに接続されているためです。
- **UpdateMode: upRefreshAll** に設定します。

```
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;
```

Dataset クライアントを設定する

新しいデータセットプロトコルクライアントを作成し、以下のプロパティを使用して設定します

- **ApplyUpdates: True** に設定すると、変更があるたびにサーバーに送信されます。
- **AutoSubscribe: True** に設定すると、新しいクライアントがサーバーに接続するたびに、クライアントは自動的に更新、削除、新しいレコードにサブスクライブします。
- **UpdateMode: upRefreshAll** に設定します。

```
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;
```

プロトコルファイル

このプロトコルは、バイナリ WebSocket トランスポートを使用してファイルを送信できます。低いメモリ使用量で大きなファイル进行处理できます。

機能

- パブリッシュ/サブスクライブメッセージパターンにより、1対多のメッセージ配信とアプリケーションの分離を実現します。
- **Acknowledgment** 送信されたメッセージの。
- ファイル配信のための **QoS** (Quality of Service) を実装します。
- 受信したファイルに対してオプションで **認証** をリクエストできます。
- **低メモリ** 使用量。

コンポーネント

[TsgcWSPServer_Files](#): サーバプロトコル Files VCL コンポーネント。

[TsgcWSPClient_Files](#): クライアントプロトコルFiles VCLコンポーネント。

Classes

[TsgcWSMessageFile](#): ファイルパケット情報をカプセル化するオブジェクト。

最も一般的な使用方法

- ファイルの送信
 - [サーバーへのファイル送信方法](#)
 - [クライアントへのファイル送信方法](#)
- 大きなファイル
 - [大きなファイルの送信方法](#)

TsgcWSPServer_Files

これはサーバーファイルプロトコルコンポーネントです。

はじめに

これはサーバーファイルプロトコルコンポーネントです。このコンポーネントをフォームに配置し、Server プロパティを使用して [TsgcWebSocketServer](#) コンポーネントを選択する必要があります。

リファレンス

- プロパティ — コンポーネントで利用可能な5つのプロパティ。
- メソッド — パブリックメソッド3件。
- イベント — 処理できる 20 のイベント。

TsgcWSPClient_Files

これはクライアントファイルプロトコルコンポーネントです。

はじめに

これはクライアントファイルプロトコルコンポーネントです。このコンポーネントをフォームに配置し、Client プロパティを使用して [TsgcWebSocketClient](#) コンポーネントを選択する必要があります。

リファレンス

- プロパティ — コンポーネントで利用可能なプロパティ5件。
- メソッド — 4個のpublicメソッド。
- イベント — 処理可能な 19 個のイベント。

TsgcWSMessageFile

このオブジェクトは、ファイルプロトコルイベントが発生するたびにパラメーターとして渡されます。

プロパティ

- BufferSize: パケットのデフォルトサイズ。
- Channel: 指定すると、このファイルは特定のチャンネルに サブスクライブしているクライアントにのみ送信されます。
- Method: 内部メソッド。
- FileId: ファイルの識別子。受信/送信されたすべてのファイルで一意です。
- Data: ユーザーカスタムデータ。ここにユーザーは任意のテキストを設定できます。
- FileName: ファイルの名前。
- FilePosition: バイト単位のファイル位置です。
- FileSize: ファイルの合計サイズ (バイト単位) 。
- Id: パケットの識別子。すべてのパケットで一意です。
- QoS: メッセージのサービス品質。
- Streaming: 内部使用のため。
- Text: 内部使用のため。

Protocol Files | How Send Files To Server

サーバーにファイルを送信するには、Files プロトコルの **SendFile** メソッドを呼び出し、**FileName** の完全パスを引数として渡します。

サーバーが受信したファイルは、デフォルトではサーバー実行可能ファイルがあるのと同じディレクトリに保存されるか、Files.SaveDirectory プロパティに設定されたパスに保存されます。

```
// ... 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

クライアントにファイルを送信するには、ファイルプロトコルの **SendFile** メソッドを呼び出し、接続の **Guid** と完全な **FileName** を引数として渡してください。クライアント接続の Guid はサーバープロトコルファイルの OnConnect イベントで取得できます。

クライアントが受信したファイルは、デフォルトでクライアント実行ファイルが存在するディレクトリ、または Files.SaveDirectory プロパティで設定されたパスに保存されます。

```
// ... 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");
```

プロトコルファイル | 大きなファイルの送信方法

サーバーまたはクライアントに大きなファイル（例: 数ギガバイトのファイル）を送信する場合、ファイル全体をロードしようとするメモリの問題が発生する可能性があります。プロトコルFilesを使用すると、ファイルをより小さなパケットで送信でき、他のピアが受信すると単一のファイルに再構築されます。 **SendFile**メソッドの **Size**パラメータを使用して、各パケットのサイズをバイト単位で設定するだけです。

```
// ... 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, "");
```


プロトコル プレゼンス

Presence プロトコルを使用すると、チャンネルにサブスクライブしているユーザーを把握できます。これにより、チャットアプリケーションの作成やオンラインユーザーの確認が容易になります（例: ゲームユーザー、チャットルーム、同じドキュメントを閲覧しているユーザーなど）。

機能

- デフォルトではユーザーは**名前**で識別されますが、これは**カスタマイズ**でき、より多くのデータを渡すことができます: email、company、twitter...
- イベント **チャンネル** の作成が許可されているか、**メンバー** が許可されているかを**承認**するために...
- 新しい **メンバー**が**参加**するたびに、すべてのメンバーに**通知**が送られます。
- **メッセージ**をパブリッシュしてすべてのチャンネル サブスクライバーに送信します。
- **低メモリ** 使用量。

コンポーネント

[TsgcWSPServer_Presence](#): サーバープロトコル Presence VCL コンポーネント。

[TsgcWSPClient_Presence](#): クライアントプロトコルプレゼンス VCL コンポーネント。

Classes

[TsgcWSPresenceMessage](#): プレゼンスパケット情報をカプセル化するオブジェクト。

TsgcWSPServer_Presence

これはサーバープレゼンスプロトコルコンポーネントです。

はじめに

これはサーバープレゼンスプロトコルコンポーネントです。このコンポーネントをフォームにドロップし、Server プロパティを使用して [TsgcWebSocketServer](#) コンポーネントを選択する必要があります。

リファレンス

- プロパティ — コンポーネントで利用可能な 6 個のプロパティ。
- メソッド — 1 つのパブリックメソッド。
- イベント — 17 個のイベントを処理できます。

TsgcWSPresenceMessage

このオブジェクトは、サーバーとクライアントのプレゼンスプロトコルで交換されるすべての内部メッセージをカプセル化します。

TsgcWSPresenceMember

ID: 内部識別子

Name: クライアントによって提供される メンバー名。

Info: クライアントが提供するメンバーの追加情報。

Data: サーバー目的で利用できる TObject。

TsgcWSPresenceMemberList

Member[i]: インデックスによるリストのメンバー

Count: リストのメンバー数

TsgcWSPresenceChannel

Name: クライアントが提供するチャンネル名。

TsgcWSPresenceMsg

Text: テキストメッセージ。クライアントが Publish メソッドを呼び出す際に提供されます

TsgcWSPresenceError

Code: エラーを識別する 整数値です

Text: エラーの説明。

TsgcWSPClient_Presence

これはクライアントプレゼンスプロトコルコンポーネントです。

はじめに

これはクライアント Presence プロトコルコンポーネントです。フォームにこのコンポーネントをドロップし、Client プロパティを使用して [TsgcWebSocketClient](#) コンポーネントを選択する必要があります。

リファレンス

- プロパティ — コンポーネントで利用可能な 7 つのプロパティ。
- メソッド — 5個のpublicメソッド。
- イベント — 処理できる 16 のイベント。

プロトコルプレゼンス JavaScript

プレゼンスプロトコル JavaScript の sgcWebSockets は **sgcWebSocket.js** と **presence.esegece.com.js** ファイルを使用します。

利用可能なメソッドを以下に示します。{%host%} と {%port%} 変数を必要に応じて置き換えてください。例: sgcWebSocket サーバーを www.example.com ウェブサイトのポート 80 でリッスンするよう設定した場合は次のように設定します:

```
<script src="http://www.example.com:80/sgcWebSockets.js"></script>
<script src="http://www.example.com:80/presence.esegece.com.js"></script>
```

接続を開く

```
<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>
```

接続後の新しいメンバー

```
<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 チャンネルをサブスクライブする

```
<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>
```

トピック1チャンネルからのアンサブスクライブ

```
<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 チャンネルにメッセージをパブリッシュする

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegece.com.js"></script>
<script>
  var socket = new sgcsws_presence('ws://{%host%}:{%port%}');
  socket.publish('Hello sgcWebSockets!', 'Topic 1');
</script>
```

メッセージを受信する

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegece.com.js"></script>
<script>
  var socket = new sgcsws_presence('ws://{%host%}:{%port%}');
  socket.on('sgcpublishmsg', function(event)
  {
    console.log('#publishmsg: ' + event.message.text);
  });
</script>
```

接続中のすべてのメンバーを取得する

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegece.com.js"></script>
<script>
  var socket = new sgcsws_presence('ws://{%host%}:{%port%}');
  socket.on('sgcgetmembers', function(event)
  {
    for (var i in event.members) {
      console.log(event.members[i].id + ' ' + event.members[i].name);
    }
  });
  socket.getmembers();
</script>
```

メンバーがサブスクライブ/サブスクライブ解除したときにアラートを表示する

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegece.com.js"></script>
<script>
  var socket = new sgcsws_presence('ws://{%host%}:{%port%}');
  socket.on('sgcnewmember', function(event)
  {
    alert('#new member: ' + event.member.name);
  });
  socket.on('sgcremovemember', function(event)
  {
    alert('#removed member: ' + event.member.name);
  });
  socket.on('sgcnewchannelmember', function(event)
  {
    alert('#new member: ' + event.member.name + ' in channel: ' + event.channel.name);
  });
  socket.on('sgcremovechannelmember', function(event)
  {
    alert('#remove member: ' + event.member.name + ' from channel: ' + event.channel.name);
  });
</script>
```

OnConnect、OnDisconnect、OnErrorイベントのアラート表示

```
<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>
```

接続を閉じる

```
<script src="http://{%host%}:{%port%}/sgcWebSockets.js"></script>
<script src="http://{%host%}:{%port%}/presence.esegece.com.js"></script>
<script>
  socket.close();
</script>
```

プロトコル E2EE

エンドツーエンド暗号化 (E2EE) は、メッセージが送信者のデバイスで暗号化され、受信者のデバイスでのみ復号化できることを意味します。サーバーはパケットをルーティングしますが、平文コンテンツを読み取ることはできません。

このトピックでは、以下の技術的なフローを説明します。

- **ダイレクトメッセージ (1:1)** 2つのピア間。
- **グループメッセージ**は送信者キーモデルとメンバーシップベースのキーローテーションを使用します。

暗号化ビルディングブロック

- **IDの確認 / 鍵合意:** Elliptic Curve Diffie-Hellman (ECDH) P-256です。
- **対称暗号化:** AES-256-GCM。
- **鍵導出:** 共有シークレットから暗号化キーを導出するための HKDF-SHA-256。
- **メッセージ認証:** AEAD 認証タグ (AES-GCM によって提供)。
- **リプレイ保護:** メッセージごとのナンス/IVと送信者シーケンスカウンター。

ダイレクトメッセージ (1:1) の技術的フロー

1. **公開鍵の発見**
各ピアは公開IDキーを公開します。サーバーは公開鍵を配布できますが、秘密鍵はクライアントデバイスから外に出ることはありません。
2. **共有シークレットの確立**
送信者と受信者は ECDH (秘密鍵 + ピアの公開鍵) を実行し、同じ共有シークレットを取得します。
3. **セッションキーの導出**
HKDF-SHA-256 は ECDH 出力から 1 つ以上の対称キー (暗号化キー、オプションのヘッダーキー) を導出します。
4. **メッセージの暗号化**
平文はユニークなナンス/IVを使用して AES-256-GCM で暗号化されます。出力には暗号文と認証タグが含まれます。
5. **トランスポート**
サーバーは平文へのアクセスなしに暗号化されたペイロードとメタデータ (例: 送信者 ID、キー ID、カウンター、タイムスタンプ) を転送します。
6. **受信者の復号化**
受信者は同じセッションキーを導出し、認証タグを検証して復号化します。改ざんがあると認証が失敗します。

グループメッセージの技術フロー (Sender Keys)

グループの場合、各メッセージをメンバーごとに個別に暗号化するのはコストがかかります。一般的な最適化は **sender key** 設計です:

- 各送信者はグループごとに**送信者キー状態**を維持します。
- 状態には対称な **チェーンキー** とメッセージカウンターが含まれます。
- 送信されるグループメッセージごとに、送信者はチェーンキーからワンタイムメッセージキーを導出し、チェーンを進めます (ハッシュラチェット)。
- メッセージキーは AES-256-GCM でペイロードを暗号化します。

送信者キーの配布方法

1. 送信者がグループに参加すると、新しい送信者キーの状態を作成します。
2. 送信者キーの状態は、既存の 1 対 1 の暗号化セッション (ペアワイズ E2EE) を通じて各現在のメンバーに共有されます。
3. 配布後、通常のグループペイロードはセnderキーの高速パス (メッセージごとに 1 回の暗号化) を使用します。

メンバーシップの変更と送信者キーのローテーション

前方秘匿性と後方秘匿性を維持するため、メンバーシップイベントではグループ送信者キーをローテーションする必要があります:

- **ユーザーがグループに参加した場合:** 新しいメンバーが古い履歴を復号化できないよう、送信者キーをローテーションします (明示的に共有しない限り)。
- **ユーザーが退出または削除された場合:** 削除されたメンバーが新しいトラフィックを復号化できないよう、すぐに送信者キーをローテーションします。
- **定期的なローテーション:** オプションの時間ベースまたはメッセージカウントのローテーションにより、キー漏洩の影響を軽減します。

典型的なローテーションシーケンス:

1. 新しいセNDERキーの状態 (新しいキーID、新しいチェーンキー) を作成します。
2. ペアワイズ E2EE チャンネルを通じて、現在認可されているメンバーのみに配布します。
3. 新しいキー ID で新しいグループメッセージの暗号化を開始します。
4. 短い移行期間中のみ古いキーIDを受け入れ、その後廃止します。

セキュリティプロパティ

- **機密性:** 有効なキーを持つクライアントのみが復号化できます。
- **整合性と真正性:** AEAD 検証により、改ざんや偽造された暗号文が検出されます。
- **サーバーの盲目性:** リレーサーバーはメッセージコンテンツの平文鍵を保持しません。
- **メンバーシップ後の保護:** ローテーションされたキーにより、元メンバーは将来のグループメッセージからブロックされます。

コンポーネント

[TsgcWSPServer E2EE](#): サーバープロトコルE2EEコンポーネントです。メッセージの内容を知ることなく、クライアント間で暗号化されたメッセージを転送します。

[TsgcWSPClient E2EE](#): クライアントプロトコル E2EE コンポーネント。ピアデバイス上でのキー交換、暗号化、復号化を管理します。

TsgcWSPServer_E2EE

E2EEサーバープロトコルコンポーネントは、ペイロードを不透明に保ちながら、クライアント間で暗号化されたダイレクトメッセージとグループメッセージをルーティングします。

はじめに

E2EEサーバープロトコルコンポーネントは、ペイロードを不透明に保ちながら、クライアント間で暗号化された直接メッセージとグループメッセージをルーティングします。フォームにドロップし、[TsgcWebSocketHTTPServer](#)（または他のサーバーコンポーネント）を**Server**プロパティに割り当て、E2EEオプションを構成します。実装はユニット**sgcWebSocket_Protocol_E2EE_Server**にあります。

設定

コンポーネントを使用するには:

- 設定済みの **TsgcWebSocketHTTPServer** インスタンスに **Server** を設定します。
- **E2EE_Options.Ack** で確認応答処理を設定します。
- **E2EE_Options.PublicKeys.ReceiveNewPublicKey** で新しい公開鍵の処理方法を選択します。
- リッスンを開始するには **Server.Active := True** を設定します。

リファレンス

- プロパティ — コンポーネントに 5 つのプロパティがあります。
- メソッド — パブリックメソッドが 0 個あります。
- イベント — 処理できる 8 つのイベント。

TsgcWSPClient_E2EE

E2EEクライアントプロトコルコンポーネントは、暗号化されたダイレクトメッセージや暗号化されたグループメッセージを含む、WebSocket接続上のエンドツーエンド暗号化を追加します。

はじめに

E2EE クライアントプロトコルコンポーネントは、暗号化されたダイレクトメッセージや暗号化されたグループメッセージを含む、WebSocket 接続上のエンドツーエンド暗号化を追加します。コンポーネントをフォームにドロップし、[TsgcWebSocketClient](#) を **Client** プロパティに割り当て、接続前に E2EE オプションを設定します。実装は **sgcWebSocket_Protocol_E2EE_Client** ユニットにあります。

設定

コンポーネントを使用するには:

- 設定済みの **TsgcWebSocketClient** インスタンスに **Client** を設定します。
- **E2EE_Options.UserId**を使用して一意のユーザー識別子を割り当てます。
- デリバリー状態のコールバックが必要な場合は、**E2EE_Options.Ack.RcvDirectMessage** と **E2EE_Options.Ack.RcvGroupMessage** でメッセージ確認を有効にします。
- **Client.Active := True** を設定して WebSocket 接続を開きます。

リファレンス

- プロパティ — コンポーネントで利用可能な5つのプロパティ。
- メソッド — 9 個のパブリックメソッド。
- イベント — 処理可能な19のイベント。

WebSocket APIs

WebSocket をベースとした実装がいくつかあります: 金融、メッセージ発行、キューなど。sgcWebSockets は WebSocket プロトコルをベースとした最も重要な API を実装しています。API を使用するには、API コンポーネントをクライアントに添付するだけで、すべてのメッセージが API コンポーネントによって処理されます（クライアントに添付できる API コンポーネントは1つのみです）。

クライアント API

API	説明
Binance	国際的な多言語暗号通貨取引所です。
Binance Futures	Binance Futures の WebSocket / REST マーケットストリームに接続できます。
Coinbase	Coinbaseは米国を拠点とする暗号資産取引所です。Bitcoin (BTC)、Ethereum (ETH) を取引します。WebSocket APIとREST APIをサポートしています。
SignalR	WebSocket をトランスポートとして使用したリアルタイム Web 機能の開発を支援するライブラリです。
SignalRCore	ASP.NET Core SignalR は、アプリにリアルタイム Web 機能を追加することを簡易化します。
SocketIO	リアルタイム Web アプリケーション向けの JavaScript ライブラリです。Web クラウド環境でリアルタイムな双方向通信を実現します。
Kraken	米国を拠点とする暗号資産取引所です。
Kraken Futures	Kraken Futures の WebSocket / REST マーケットデータに接続できます。
Pusher	Pusher は、柔軟な pub/sub メッセージング、ライブユーザーリスト、認証などを提供するクラウドプラットフォームです。
Bitfinex	Bitfinex は世界最大かつ最も高度な暗号通貨取引プラットフォームの1つです。Bitcoin、Ethereum、Ripple、EOS、Bitcoin Cash、Iota、NEO、Litecoin、Ethereum Classic などを取引します。
Bitstamp	Bitstampは、2011年からブロックチェーンエコシステムをサポートしている、世界的な暗号資産取引所の1つです。
Huobi	国際的な多言語暗号通貨取引所です。
Cex	は暗号通貨取引所であり、かつてのBitcoinクラウドマイニングプロバイダーです。
Cex Plus	CEX.IO Exchange Plus は、深い流動性と高度な取引ツールを備えた究極の仮想通貨取引プラットフォームです。
Bitmex	は暗号通貨取引所およびデリバティブ取引プラットフォームです。
3Commas	暗号通貨取引ロボットです。
Kucoin	は、BTC、ETH、KCS、SHIB、DOGEなどの暗号通貨を売買・保管できる暗号通貨取引プラットフォームです。
Kucoin Futures	Kucoin フューチャーズサーバー（WebSocket および REST）への接続を可能にします。
OKX	旧称 OKEx。暗号通貨スポットおよびデリバティブ取引所の最大手の1つです。
XTB	FX および CFD 取引。2000 以上の金融市場へのアクセスを提供します。
Discord	は、オンラインゲームやストリーミングで最も人気のあるコミュニケーションツールです。
Bybit	暗号通貨取引所および取引プラットフォーム
OpenAI API	OpenAI Realtime API は、音声間の会話体験やリアルタイムの文字起こしを含む、リアルタイムのアプリケーションを実現します。
MEXC	集中型暗号通貨取引所および取引プラットフォーム。このコンポーネントは SPOT 取引をサポートします。
MEXC Futures	中央集権型の暗号通貨取引所および取引プラットフォームです。このコンポーネントは FUTURES 取引をサポートします。
Bitget	スポットおよび先物市場をサポートする暗号通貨取引所および取引プラットフォーム
Gate.io	WebSocket および REST API を使用したスポットおよび先物取引をサポートする暗号通貨取引プラットフォーム
Deribit	BitcoinとEthereumの先物およびオプション取引を提供する暗号通貨デリバティブ取引プラットフォーム
Crypto.com	WebSocket および REST API を使用した Market チャンネルと User チャンネルを提供する暗号通貨取引プラットフォーム

WebSocket API は**実行時**に登録できます。メソッド **RegisterAPI** を呼び出し、API コンポーネントをパラメータとして渡します。

Other Client APIs

API	説明
Telegram	クラウドベースのインスタントメッセージングおよび VoIP サービスです。ユーザーオ、ステッカー、音声、あらゆる種類のファイルを交換できます。
Whatsapp	国際的に利用可能なアメリカのフリーウェアで、クロスプラットフォームの集中型イボイスオーバー IP サービスです
RCON	サーバーにコンソールコマンドを発行できる TCP/IP ベースの通信プロトコルです。
CryptoHopper	暗号取引ボットおよびポートフォリオマネージャーです。
CryptoRobotics	仮想通貨の取引ロボットです。

サーバー API

API	説明
RTCMultiConnection	RTCMultiConnection は、ピアツーピアアプリケーション（画面共有、音声、ビデオストリーミングなど）向けの WebRTC JavaScript ライブラリです。
WebPush	Web Push プロトコルにより、ウェブアプリケーションはアプリケーションのい場合でもユーザーに通知を送信できます。
WebAuthn	生体認証、セキュリティキー、パスキーを使用したパスワードレス認証の API。

API Binance

Binance

Binanceは国際的な多言語の暗号通貨取引所です。BinanceのデータにアクセスするためのいくつかのAPIを提供しています。以下のAPIがサポートされています:

1. **WebSocketストリーム**: 一部のメソッドを購読してリアルタイムでデータを取得できます。イベントはサーバーから購読者であるクライアントにプッシュされます。プロトコルとしてWebSocketを使用します。
2. **UserData ストリーム**: サブスクライブしたクライアントはアカウントの詳細を受け取ります。認証にはAPI キーが必要で、WebSocket をプロトコルとして使用します。
3. **REST API**: 認証にAPI キーとシークレットが必要で、プロトコルとして HTTPS を使用します。
 1. [マーケットデータ](#)
 2. [アカウントおよび取引データ](#)
 3. [ウォレット](#)
4. **Futures**: WebSocket 先物市場データストリームは [Binance Futures Client API](#) を通じてサポートされています。

クライアントは **Binance.us** もサポートしており、次のAPI がサポートされています:

1. **WebSocketストリーム**: 一部のメソッドを購読してリアルタイムでデータを取得できます。イベントはサーバーから購読者であるクライアントにプッシュされます。プロトコルとしてWebSocketを使用します。
2. **UserData ストリーム**: サブスクライブしたクライアントはアカウントの詳細を受け取ります。認証にはAPI キーが必要で、WebSocket をプロトコルとして使用します。
3. **REST API**: クライアントはサーバーの市場データやアカウントデータをリクエストできます。認証にはAPI キーとシークレットが必要で、プロトコルとして HTTPS を使用します。

プロパティ

Binance API には public と private の2種類のメソッドがあります。パブリックメソッドはティッカー価格の取得など、認証なしでアクセスできます。一部はプライベートであり、ユーザーデータに関連するため、Binance API キーの使用が必要です。

- **ApiKey**: Binance アカウントで新しいAPI キーをリクエストできます。その値をこのプロパティにコピーしてください。
- **ApiSecret**: API シークレットは REST_API のみ必要です。WebSocket API は一部のメソッドのみ ApiKey が必要です。
- **TestNet**: 有効にするとBinanceデモアカウントに接続します (デフォルトはfalse)。
 - **HTTPLogOptions**: HTTPリクエストのログをテキストファイルに保存します
 - **Enabled**: 有効にすると、WebSocket APIのすべてのHTTPリクエストを保存します。
 - **FileName**: ログを保存するファイルのフルパス
 - **REST**: REST APIリクエストのログをテキストファイルに保存します
 - **Enabled**: 有効にすると、REST APIのすべてのHTTPリクエストを保存します。
 - **FileName**: ログを保存するファイルのフルパス。
- **UserStream**: 有効にすると、クライアントはアカウント、注文、または残高の更新に関する通知を受け取ります (デフォルトはtrue)。
- **BinanceUS**: 有効にすると、Binance.us サーバーに接続します (デフォルトの Binance.com サーバーの代わりに)。
- **ListenKeyOnDisconnect**: このプロパティは、アクティブな ListenKey を持つ Binance サーバーからクライアントが切断されたときの動作を指定します。
 - **blkodDeleteListenKey**: Binance サーバーへ HTTP リクエストを行ってアクティブな ListenKey を削除します (これがデフォルトです)。
 - **blkodClearListenKey**: Binance サーバーから ListenKey を削除せず、フィールドの値をクリアするだけです。
 - **blkodDoNothing**: 何も行いません。次に Binance に接続するときに同じ ListenKey を使用しようとしています。
- **UseCombinedStreams**: 有効にすると、ストリームを次のように結合します:
 {"stream": "<streamName>", "data": <rawPayload>} (デフォルトでは無効)

最も一般的な使用方法

- **WebSockets API**
 - [WebSocket API への接続方法](#)
 - [WebSocket チャンネルへのサブスクライブ方法](#)
- **REST API**
 - [マーケットデータの取得方法](#)
 - [プライベート REST API の使用方法](#)
 - [スポット取引の方法](#)
 - [プライベートリクエストの時間](#)
 - [出金](#)

WebSocket Stream API

ベースエンドポイントは `wss://stream.binance.com:9443` です。クライアントは接続成功後にイベントをサブスクライブ/アンサブスクライブできます。
以下のサブスクリプション/サブスクリプション解除メソッドがサポートされています。

メソッド	パラメータ	説明
AggregateTrades	銘柄	単一のテイカー注文に集約された取引情報をプッシュします
取引	銘柄	生の取引情報をプッシュします。各取引には固有の買い手と売り手がいます
KLine	Symbol, Interval	現在の klines/ローソク足を毎秒、毎分、毎時間プッシュ更新します。
MiniTicker	銘柄	24時間ローリングウィンドウのミニティッカー統計です。これらはUTC日の統計ではなく、過去24時間の24時間ローリングウィンドウです。
AllMiniTickers		配列内で変更されたすべてのシンボルの 24 時間ローリングウィンドウのミニティッカー統計。これらは UTC 日の統計ではなく、過去 24 時間の 24 時間ローリングウィンドウです。変更されたティッカーのみが配列に存在することに注意してください。
Ticker	銘柄	単一銘柄の24時間ローリングウィンドウのティッカー統計。これらはUTC日の統計ではなく、過去24時間の24時間ローリングウィンドウです。
AllMarketTickers		変動があったすべての銘柄の24時間ローリングウィンドウのティッカー統計（配列形式）。これらはUTC日の統計ではなく、過去24時間の24時間ローリングウィンドウです。変動のあったティッカーのみが配列に含まれます。
BookTicker	銘柄	指定されたシンボルについて、最良買い気配または売り気配の価格や数量への更新をリアルタイムでプッシュします。
AllBookTickers		すべての銘柄の最良の売値または買値の価格または数量の更新をリアルタイムでプッシュします。
PartialBookDepth	Symbol, Depth	上位 <levels> 件の買いと売りを毎秒プッシュします。有効な <levels> は 5、10、または 20 です。
DiffDepth	銘柄	ローカルで注文板を管理するために使用される注文板の価格と数量の深度更新。

サブスクライブ/アンサブスクライブが成功した後、クライアントはその結果に関するメッセージを受け取ります。id は Subscribed / Unsubscribed メソッドの結果です。

```
{
  "result": null,
```



```

    "id": 1
  }

```

User Data Stream API

Binanceアカウントから取得した有効なApiKeyが必要で、ApiKey はコンポーネントの `Binance.ApiKey` プロパティに設定する必要があります。

変化があるたびに次のデータがクライアントにプッシュされます。任意のメソッドをサブスクライブする必要はありません。有効な ApiKey が設定されていれば自動的に行われます。

メソッド	説明
アカウントのアップデート	アカウントの状態は <code>outboundAccountInfo</code> イベントで更新されます。
残高更新	残高の更新は以下の場合に発生します： - 口座への入金または出金 - 口座間の資金移動（例：スポットからマージンへ）
注文の更新	注文は <code>executionReport</code> イベントで更新されます。

REST API

ベースエンドポイントは `https://api.binance.com` です。すべてのエンドポイントは JSON オブジェクトまたは配列を返します。データは昇順で返されます。最古が最初、最新が最後です。

プロパティ `REST_API.BinanceOptions` を使用して REST API オプションにアクセスします。

パブリック API エンドポイント

これらのエンドポイントには認証なしでアクセスできます。

汎用エンドポイント

メソッド	パラメータ	説明
Ping		REST API への接続をテストします。
GetServerTime		REST API への接続をテストして現在のサーバー時刻を取得します。
GetExchangeInformation		現在の取引所の取引ルールとシンボル情報

Market Data エンドポイント

メソッド	パラメータ	説明
GetOrderBook	銘柄	オーダーブックを取得します。
GetTrades	銘柄	最近の取引を取得します。
GetHistoricalTrades	銘柄	古い取引を取得します。

GetAggregateTrades	銘柄	圧縮された集約トレードを取得します。同じ注文から、同じ価格で、同時に約定したトレードは数量が集計されます。
GetKLines	Symbol, Interval	銘柄の kline/ローソク足バー。Klines は開始時刻で一意に識別されます。
GetAveragePrice	銘柄	シンボルの現在の平均価格。
Get24hrTicker	銘柄	24 時間のローリングウィンドウの価格変動統計。シンボルなしでアクセスする場合は注意が必要です。
GetPriceTicker	銘柄	シンボルの最新価格。
GetPriceTickers	シンボル	シンボルの配列の最新価格。例: ["BTCUSDT", "BNBUSDT"]
GetBookTicker	銘柄	シンボルまたは複数シンボルの板注文における最良価格/数量。
GetUIKLines	Symbol, Interval	シンボルのKライン/ローソク足バー。レスポンスは GetKLines に似ており、ローソク足チャートの表示に最適化されています。
GetRollingWindowTicker	Symbol, Symbols, WindowSize	ローリングウィンドウの価格変動統計。注意: WindowSize のデフォルトは、指定されない場合 1d です。
GetTradingDayTicker	Symbol, Symbols, Type	取引日の価格変動統計。

プライベート API エンドポイント

サーバーによる認証を取得するために APIKey と APISecret が必要です。

アカウントデータエンドポイント

メソッド	パラメータ	説明
NewOrder	Symbol, Side, Type	新しい注文を送信します。
PlaceMarketOrder	Side, Symbol, Quantity	新しい成行注文を発注します。
PlaceMarketQuoteOrder	Side, Symbol, QuoteOrderQty	新規マーケットクオート注文を発注します
PlaceLimitOrder	Side, Symbol, Quantity, LimitPrice	新規指値注文を発注します
PlaceStopOrder	Side, Symbol, Quantity, StopPrice, LimitPrice	新しいストップ注文を発注します。
PlaceStopTrailingOrder	Side, Symbol, Quantity, TrailingDelta, LimitPrice	新しいトレーリングストップ注文を出します。
PlaceTakeProfitOrder	Side, Symbol, Quantity, StopPrice, LimitPrice	新しい利益確定注文を発注します。
PlaceTakeProfitTrailingOrder	Side, Symbol, Quantity, TrailingDelta, LimitPrice	新しいTake ProfitのTrailing注文を発注します
PlaceLimitMakerOrder	Side, Symbol, Quantity	新しいLimit Market Orderを発注します
TestNewOrder	Symbol, Side, Type	新しい注文の作成と署名/recvWindow の長さをテストします。新しい注文を作成して検証しますが、マッチングエンジンには送信しません。
QueryOrder	銘柄	注文のステータスを確認します。

CancelOrder	銘柄	アクティブな注文をキャンセルします。OrderId または OrigClientOrderId のいずれかを送信する必要があります。
CancelAllOpenOrders	シンボル（オプション）	
GetOpenOrders		シンボルのすべてのオープン注文を取得します。シンボルなしでアクセスする場合は注意してください。
GetAllOrders	銘柄	すべてのアカウント注文（アクティブ、キャンセル済み、または約定済み）を取得します。
NewOCO	Symbol, Side, Quantity, Price, StopPrice	新しいOCO注文の送信
CancelOCO	銘柄	注文リスト全体をキャンセルする
QueryOCO	銘柄	提供されたオプションのパラメータに基づいて特定のOCOを取得します
GetAllOCO		指定されたオプションパラメータに基づいてすべてのOCOを取得します
GetOpenOCO		すべてのオープンOCOを取得します。
GetAccountInformation		現在のアカウント情報を取得します。
GetAccountTradeList	銘柄	特定のアカウントとシンボルの取引を取得します。
CancelReplaceOrder	Symbol, Side, Type, CancelReplaceMode	既存の注文をキャンセルして同じシンボルで新しい注文を発注します。
NewOrderListOCO	Symbol, Side, Quantity, AboveType, BelowType	新しいOCO注文リストを発注します。
NewOrderListOTO	Symbol, WorkingType, WorkingSide, WorkingQuantity, WorkingPrice, PendingType, PendingSide, PendingQuantity	新しいOTO（One-Triggers-the-Other）注文リストを発注します。
NewOrderListOTOCO	Symbol, WorkingType, WorkingSide, WorkingQuantity, WorkingPrice, PendingSide, PendingAboveType, PendingBelowType, PendingQuantity	新しいOTOCO（One-Triggers-a-One-Cancels-the-Other）注文リストを発注します。
NewSOROrder	Symbol, Side, Type, Quantity	スマートオーダールーティング（SOR）を使用して注文を出します。
TestSOROrder	Symbol, Side, Type, Quantity	Smart Order Routing（SOR）を使用して新しい注文をテストします。新しい注文を作成して検証しますが、マッチングエンジンには送信しません。
GetOrderRateLimitUsage		全インターバルのユーザーの現在の注文数使用量を表示します。
GetPreventedMatches	銘柄	STP（自己取引防止）により期限切れになった注文の一覧を表示します。
GetAllocations	銘柄	SOR注文配置から生じるアロケーションを取得します。
GetAccountCommission	銘柄	現在のアカウント手数料率を取得します。

エンドポイントの変換

メソッド	パラメータ	説明
GetAllConvertPairs	FromAsset, ToAsset	すべての変換可能なトークンペアと、それぞれのトークンの上限/下限を照会します
GetConvertAssetInfo		サポートされている資産の精度情報を照会します
SendConvertQuoteRequest	FromAsset, ToAsset	リクエストされたトークンペアの見積もりをリクエストします。
AcceptConvertQuote	QuoteId	クォート ID でオファーされたクォートを承認します。
GetConvertOrderStatus	OrderId または QuoteId	注文 ID で注文ステータスを照会します。
PlaceConvertLimitOrder	BaseAsset, QuoteAsset, Side, LimitPrice	ユーザーが指値注文を発注できるようにします。baseAsset または quoteAsset は exchangeInfo エンドポイントで確認できます。Limit 価格は baseAsset から quoteAsset の方向で定義されます。baseAmount または quoteAmount のいずれかを使用します。
CancelConvertLimitOrder	OrderId	ユーザーが指値注文をキャンセルできるようにします
GetConvertLimitOpenOrders		既存のすべての指値注文を照会できるようにします
GetConvertTradeHistory	StartTime、EndTime	startTimeとendTimeの最大間隔は30日です。

ウォレットエンドポイント

(*ウォレットエンドポイントはデモではなく本番サーバーでのみ動作します)

メソッド	説明
GetWalletSystemStatus	システムステータスを取得します。
GetWalletAllCoinsInformation	ユーザーのコイン情報を取得します (入出金可能なもの)。
GetWalletDailyAccountSnapshot	Type: "SPOT", "MARGIN", "FUTURES" - クエリの期間は30日未満である必要があります - 過去1か月以内のクエリのみをサポートします - startTimeとendTimeが送信されない場合、デフォルトで過去7日間のレコードを返します
SetWalletDisableFastWithdrawSwitch	このリクエストはアカウントの即時出金スイッチを無効にします。 このエンドポイントをリクエストするAPIキーの「trade」オプションを有効にする必要があります。
SetWalletEnableFastWithdrawSwitch	このリクエストはアカウントの高速出金スイッチを有効にします。 このエンドポイントをリクエストするAPIキーに「trade」オプションを有効にする必要があります。 高速出金スイッチがオンになると、Binance アカウントへの資金移動が即時に行われます。オンチェーントランザクション、トランザクション ID、出金手数料は発生しません。
WalletWithdraw	出金リクエストを送信します。
GetWalletDepositHistory	入金履歴を取得します。
GetWalletWithdrawHistory	出金履歴を取得します。

GetWalletDepositAddress	ネットワークを指定して入金アドレスを取得します。
GetWalletAccountStatus	アカウントステータスの詳細を取得します。
GetWalletAccountAPITradingStatus	アカウントのAPI取引ステータスの詳細を取得します。
GetWalletDustLog	最後の 100 件のレコードのみを返す 2020/12/01 以降のレコードのみを返す
GetWalletAssetsConvertedBNB	
WalletDustTransfer	ダスト資産をBNBに変換します。 このエンドポイントにリクエストする API Key のスポット & マー ジン取引権限を有効にする必要があります。
GetWalletAssetDividendRecord	資産の配当記録を照会します。
GetWalletAssetDetail	Binance でサポートされているアセットの詳細を取得します。
GetWalletTradeFee	取引手数料を取得します
WalletUserUniversalTransfer	このエンドポイントにリクエストする API キーに対してユニバー サル転送の許可オプションを有効にする必要があります。 MAIN_UMFUTURE スポットアカウントから USD⑤-M 先物アカ ウントへの振替 タイプの列挙型: - MAIN_CMFUTURE スポット口座から COIN-M 先物口座へ の振替 - MAIN_MARGIN スポット口座からマージン（クロス）口座 への振替 - UMFUTURE_MAIN USD⑤-M 先物口座からスポット口座へ の振替 - UMFUTURE_MARGIN USD⑤-M 先物口座からマージン （クロス）口座への振替 - CMFUTURE_MAIN COIN-M 先物口座からスポット口座へ の振替 - CMFUTURE_MARGIN COIN-M 先物口座からマージン（ク ロス）口座への振替 - MARGIN_MAIN マージン（クロス）口座からスポット口座 への振替 - MARGIN_UMFUTURE マージン（クロス）口座から USD⑤-M 先物への振替 - MARGIN_CMFUTURE マージン（クロス）口座から COIN- M 先物への振替 - ISOLATEDMARGIN_MARGIN 分離マージン口座からマージ ン（クロス）口座への振替 - MARGIN_ISOLATEDMARGIN マージン（クロス）口座から 分離マージン口座への振替 - ISOLATEDMARGIN_ISOLATEDMARGIN 分離マージン口座 から分離マージン口座への振替 - MAIN_FUNDING スポット口座からファンディング口座へ の振替 - FUNDING_MAIN ファンディング口座からスポット口座へ の振替 - FUNDING_UMFUTURE ファンディング口座から UMFUTURE 口座への振替 - UMFUTURE_FUNDING UMFUTURE 口座からファンディ ング口座への振替 - MARGIN_FUNDING MARGIN 口座からファンディング口座 への振替 - FUNDING_MARGIN ファンディング口座からマージン口座 への振替 - FUNDING_CMFUTURE ファンディング口座から CMFUTURE 口座への振替 - CMFUTURE_FUNDING CMFUTURE 口座からファンディ ング口座への振替
GetWalletQueryUserUniversalTransferHistory	type が ISOLATEDMARGIN_MARGIN および ISOLATEDMARGIN_ISOLATEDMARGIN の場合、 fromSymbol を送信する必要があります。

	• type が MARGIN_ISOLATEDMARGIN および ISOLATEDMARGIN_ISOLATEDMARGIN の場合、toSymbol を送信する必要があります。 • 過去 6 ヶ月以内のクエリのみサポートされます。 • startTime と endTime が送信されない場合、デフォルトで過去 7 日間のレコードを返します。
GetWalletFundingWallet	現在、以下のビジネス資産のクエリをサポートしています: Binance Pay、Binance Card、Binance Gift Card、Stock Token
GetWalletUserAsset	ユーザー資産を取得します（プラスデータのみ）。
GetWalletApiKeyPermission	

イベント

Binance のメッセージは TsgcWebSocketClient コンポーネントで受信されます。以下のイベントを使用できます。

OnConnect

Binance サーバーへの接続が成功した後。

OnDisconnect

Binance サーバーからの切断後

OnMessage

サーバーからクライアントに送信されたメッセージは、このイベントで処理されます。

OnError

プロトコルにエラーがある場合、このイベントが呼び出されます。

OnException

未処理の例外がある場合、このイベントが呼び出されます。

さらに、Binance APIコンポーネントには **OnBinanceHTTPException** という特定のイベントがあります。これは HTTPリクエスト（REST APIまたはWebSocketユーザーストリーム）の呼び出しにエラーが発生するたびに発生します。

(*) Binanceサーバーの変更により、Rad Studio 10.1より前のIndyバージョンはテストサーバーに接続できません。この問題は、Enterpriseエディションには影響せず、Indyバージョンが最新にアップグレードされている場合も影響しません。

Binance | WebSocket API への接続

Binance WebSocket APIに接続するには、新しいBinance APIクライアントを作成してTsgcWebSocketClientにアタッチするだけです。
以下の例をご覧ください:

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient(NULL);  
TsgcWSAPI_Binance *oBinance = new TsgcWSAPI_Binance(NULL);  
oBinance->Client = oClient;  
oClient->Active = true;
```

Binance | WebSocket チャンネルのサブスクライブ

Binance は、マーケットデータ、注文などのリアルタイムアップデートを受け取るために購読できるさまざまなチャンネルを提供しています。以下にティッカーへの購読方法のサンプルを示します。

```
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

Binance は REST エンドポイントを通じてパブリックなマーケットデータを提供します。これらのエンドポイントを呼び出すと、リクエストされたマーケットデータのスナップショットが返されます。マーケットデータエンドポイントは認証を必要としないため、すべてのユーザーが自由に利用できます。

使用例: ティッカー BNBBTC のスナップショットを取得するには、以下の呼び出しを行います:

```
TsgcWSAPI_Binance *oBinance = new TsgcWSAPI_Binance(this);  
ShowMessage(oBinance->REST_API->GetPriceTicker("BNBBTC"));
```


Binance | プライベート REST API

Binance REST API は公開エンドポイントと非公開エンドポイントを提供します。非公開エンドポイントでは、トランザクションのセキュリティを高めるためにメッセージへの署名が必要です。

まず Binance アカウントにログインして新しい API を作成してください。以下の値が得られます。

- ApiKey
- ApiSecret

これらのフィールドは、Binance API クライアントコンポーネントの Binance プロパティで設定する必要があります。

設定が完了したら、Binance Pro REST API へのプライベートリクエストを開始できます

*プライベートリクエストでは、ローカルマシンのローカル時刻が同期されている必要があります。同期されていない場合、リクエストはBinanceサーバーによって拒否されます。これについては、次の記事を確認してください。[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 | スポット取引

Binance では REST API を使用してスポット取引ができます。

設定

まず、binanceアカウントで**API Key**を作成し、Spotでの取引権限を追加する必要があります。

これが完了すると、現物取引を開始できます。

まず、**ApiKey** と **ApiSecret** を Binance クライアントコンポーネントに設定します。これらは Binance サーバーに送信されるリクエストの署名に使用されます。

注文を発注する

新しい注文を発注するには、Binance クライアントコンポーネントの **REST_API.NewOrder** メソッドを呼び出してください。

注文のタイプ（成行、指値など）に応じて、APIはより多くまたはより少ないフィールドを必要とします。

Mandatory Fields

- **Symbol:** 製品 ID シンボル。例: BNBBTC
- **Side:** BUY または SELL
- **type:** 注文タイプ
 - LIMIT
 - MARKET
 - STOP_LOSS
 - STOP_LOSS_LIMIT
 - TAKE_PROFIT
 - TAKE_PROFIT_LIMIT
 - LIMIT_MAKER

タイプに基づく追加の必須フィールド

- **LIMIT:** timeInForce, quantity, price
- **MARKET:** 数量または quoteOrderQty
- **STOP_LOSS / TAKE_PROFIT:** 数量、ストップ価格
- **STOP_LOSS_LIMIT / TAKE_PROFIT_LIMIT:** timeInForce, quantity, price, stopPrice
- **LIMIT_MAKER:** 数量、価格

注文を送信する際、2つの可能性があります。

1. **成功:** NewOrder 関数は Binance サーバーから送信されたメッセージを返します。
2. **エラー:** 例外は OnBinanceHttpException イベントで返されます。

BNBBTC の成行注文を 1 件発注

```
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));
```

0.009260 で BNBBTC の指値注文を 1 件発注

```
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 | プライベートリクエスト時刻

Binanceにプライベートリクエストを行う際、リクエストのセキュリティを高めるためにメッセージは署名されます。メッセージはローカル時刻を取得して署名済みメッセージ内に送信します。ローカル時刻がBinanceサーバーと5秒を超える差がある場合、リクエストは拒否されます。そのため、ローカル時刻が同期されていることを確認することが重要です。これは、OSの時刻同期メソッドを使用して行うことができます。

ロジックは以下のとおりです:

```
if (timestamp < (serverTime + 1000) && (serverTime - timestamp) <= recvWindow) {  
  // process request  
} else {  
  // reject request  
}
```

recvWindow は 5000 以下の小さい値を使用することをお勧めします。最大値は 60000 ミリ秒を超えることはできません。

GetServerTimeメソッドを呼び出すことでBinanceサーバーの時刻を確認できます。これはBinanceサーバーの時刻を返します

RecvWindow のデフォルト値は **5000** です。この値はプロパティ **REST_API.BinanceOptions.RecvWindow** を使用して増やすことができます。

Binance | 出金

Binance では Wallet API を使用して出金リクエストを送信できます。以下のパラメータのみが必須です:

- コイン
- アドレス
- 金額

```
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

Binance は国際的な多言語対応の暗号通貨取引所です。Binance のデータにアクセスするための API を提供しています。このコンポーネントを使用すると、Binance Futures WebSocket マーケットストリームを取得できます。

<https://binance-docs.github.io/apidocs/futures/en>
<https://binance-docs.github.io/apidocs/delivery/en>

先物契約

Binance API には 2 種類のメソッドがあります: パブリックとプライベートです。パブリックメソッドは認証なしでアクセスできます (例: ティッカー価格の取得)。一部はプライベートでユーザーデータに関連しており、これらのメソッドは Binance API キーの使用が必要です。

- **ApiKey**: Binance アカウントで新しい API キーをリクエストできます。その値をこのプロパティにコピーしてください。
- **ApiSecret**: API シークレットは REST_API のみ必要です。WebSocket API は一部のメソッドのみ ApiKey が必要です。
- **TestNet**: 有効にすると Binance デモアカウントに接続します (デフォルトは false)。
 - **HTTPLogOptions**: HTTP リクエストのログをテキストファイルに保存します
 - **Enabled**: 有効にすると、WebSocket API のすべての HTTP リクエストを保存します。
 - **FileName**: ログを保存するファイルのフルパス
 - **REST**: REST API リクエストのログをテキストファイルに保存します
 - **Enabled**: 有効にすると、REST API のすべての HTTP リクエストを保存します。
 - **FileName**: ログを保存するファイルのフルパス。
- **UserStream**: 有効にすると、クライアントはアカウント、注文、または残高の更新に関する通知を受け取ります (デフォルトは true)。
- **ListenKeyOnDisconnect**: このプロパティは、アクティブな ListenKey がある状態でクライアントが Binance サーバーから切断した場合の動作を指定します。
 - **blkodDeleteListenKey**: Binance サーバーへの HTTP リクエストによりアクティブな ListenKey を削除します (これがデフォルトです)。
 - **blkodClearListenKey**: Binance サーバーから ListenKey を削除せず、フィールドの値のみをクリアします。
 - **blkodDoNothing**: 何もしないため、次回 Binance に接続する際に同じ ListenKey を使用しようとしています。
- **UseCombinedStreams**: 有効にすると、ストリームを次のように結合します:
 {"stream": "<streamName>", "data": <rawPayload>} (デフォルトでは無効)

クライアントは **USDT** または **COIN** Binance Futures に接続できます。**FuturesContracts** プロパティを使用して取引するコントラクトを設定します:

- **bfcUSDT**: USD-M 先物 API に接続します。
- **bfcCOIN**: COIN-M Futures API に接続します。

クライアントは Production または Demo の Binance アカウントに接続できます。**TestNet** プロパティが有効な場合は Demo アカウントに接続し、そうでない場合は本番の Binance サーバーに接続します。

WebSocket Stream API

クライアントは接続が成功した後、イベントのサブスクライブ/サブスクライブ解除ができます。以下のサブスクリプション/サブスクリプション解除メソッドがサポートされています。

メソッド	パラメータ	説明

AggregateTrades	銘柄	アグリゲートトレードストリームは、100ミリ秒ごとに単一のテイクオーダー注文のトレード情報を集約してプッシュします。
MarkPrice	Symbol, UpdateSpeed	単一シンボルのマーク価格とファンディングレートを 3 秒ごとまたは毎秒プッシュします。
AllMarkPrice	UpdateSpeed	すべてのシンボルのマーク価格とファンディングレートが 3 秒ごとまたは毎秒プッシュされます。
KLine	Symbol, Interval	Kline/Candlestick Streamは、現在のkline/ローソク足の更新を250ミリ秒ごとにプッシュします（存在する場合）。
MiniTicker	銘柄	単一シンボルの 24 時間ローリングウィンドウのミニティッカー統計。これらは UTC 日の統計ではなく、requestTime から 24 時間前までの 24 時間ローリングウィンドウです。
AllMiniTicker		すべてのシンボルの 24 時間ローリングウィンドウミニティッカー統計。これは UTC 日の統計ではなく、requestTime から 24 時間前までの 24 時間ローリングウィンドウです。変更があったティッカーのみが配列に含まれます。
Ticker	銘柄	単一シンボルの 24 時間ローリングウィンドウティッカー統計。UTC 日付の統計ではなく、requestTime から 24 時間前までの 24 時間ローリングウィンドウです。
AllMarketTickers		すべてのシンボルの 24 時間ローリングウィンドウのティッカー統計。これらは UTC の日次統計ではなく、requestTime から 24 時間前までの 24 時間ローリングウィンドウです。変化したティッカーのみが配列に含まれることに注意してください。
BookTicker	銘柄	指定されたシンボルについて、最良買い気配または売り気配の価格や数量への更新をリアルタイムでプッシュします。
AllBookTickers		すべての銘柄の最良の売値または買値の価格または数量の更新をリアルタイムでプッシュします。
LiquidationOrders	銘柄	Liquidation Order Streams は、特定の銘柄の強制清算注文情報をプッシュします
AllLiquidationOrders		全清算注文ストリームは、市場のすべてのシンボルの強制清算注文情報をプッシュします。
PartialBookDepth	Symbol、Depth	上位の買い注文と売り注文。有効値は5、10、または20です。
DiffDepth	銘柄	買い気配と売り気配。250ミリ秒、500ミリ秒、100ミリ秒ごと、またはリアルタイムでプッシュされます（存在する場合）

サブスクリプション/サブスクリプション解除が成功すると、クライアントはそれに関するメッセージを受信します。ここでidはSubscribed / Unsubscribedメソッドの結果です。

```
{
  "result": null,
  "id": 1
}
```

User Data Stream API

Binanceアカウントから取得した有効なApiKeyが必要で、ApiKey はコンポーネントの Binance.ApiKey プロパティに設定する必要があります。

変化があるたびに次のデータがクライアントにプッシュされます。任意のメソッドをサブスクライブする必要はありません。有効な ApiKey が設定されていれば自動的に行われます。

メソッド	説明

マージンコール	ユーザーのポジションリスク比率が高すぎる場合、このストリームがプッシュされます。このメッセージはリスクガイダンス情報としてのみ使用され、投資戦略としての使用は推奨されません。市場が非常に変動した場合、このストリームがプッシュされると同時にユーザーのポジションが強制決済されている可能性があります。
残高およびポジションの更新	残高の更新は以下の場合に発生します： - 残高またはポジションが更新されると、このイベントがプッシュされます。 「FUNDING FEE」がユーザーの残高に変更を加えるとき。
注文の更新	新しい注文が作成されたとき、注文ステータスが変更されるとこのようなイベントがプッシュされます。

REST API

すべてのエンドポイントは JSON オブジェクトまたは配列を返します。データは昇順 (古いものから新しいもの) で返されます。

パブリック API エンドポイント

これらのエンドポイントには認証なしでアクセスできます。

一般エンドポイント

メソッド	パラメータ	説明
Ping		REST API への接続をテストします。
GetServerTime		REST API への接続をテストして現在のサーバー時刻を取得します。
GetExchangeInformation		現在の取引所の取引ルールとシンボル情報

Market Data エンドポイント

メソッド	パラメータ	説明
GetOrderBook	銘柄	オーダーブックを取得します。
GetTrades	銘柄	最近の取引を取得します。
GetHistoricalTrades	銘柄	古い取引を取得します。
GetAggregateTrades	銘柄	圧縮された集約トレードを取得します。同じ注文から、同じ価格で、同時に約定したトレードは数量が集計されます。
GetKLines	Symbol, Interval	銘柄の kline/ローソク足バー。Klines は開始時刻で一意に識別されます。
Get24hrTicker	銘柄	24 時間のローリングウィンドウの価格変動統計。シンボルなしでアクセスする場合は注意が必要です。
GetPriceTicker	銘柄	シンボルまたは複数のシンボルの最新価格。
GetBookTicker	銘柄	シンボルまたは複数シンボルの板注文における最良価格/数量。
GetMarkPrice	銘柄	マーク価格とファンディングレート
GetFundingRateHistory	銘柄	
GetOpenInterest	銘柄	特定シンボルの現在の建玉を取得します。

GetOpenInterestStatistics	Symbol, Period	
GetTopTraderAccountRatio	Symbol, Period	
GetTopTraderPositionRatio	Symbol, Period	
GetGlobalAccountRatio	Symbol, Period	
GetTakerVolume	Symbol, Period	
GetContinuousKLines	Pair、ContractType、Interval	特定の契約タイプのKline/ローソク足バー。
GetIndexPriceKLines	Pair, Interval	ペアのインデックス価格のローソク足バー。
GetMarkPriceKLines	Symbol, Interval	シンボルのマーク価格のローソク足/Kline バー。
GetPremiumIndexKLines	Symbol, Interval	シンボルのプレミアムインデックスのローソク足バー。
GetFundingInfo		すべてのシンボルのファンディングレート情報を取得します。
GetPriceTickerV2	銘柄	シンボルまたは複数シンボルの最新価格 (V2)。
GetIndexInfo	銘柄	インデックス情報を取得します。
GetAssetIndex	銘柄	マルチアセットモードのアセットインデックスを取得します。
GetConstituents	銘柄	インデックスの構成銘柄を取得します。
GetDeliveryPrice	ペア	配達価格を取得します。
GetBasis	Pair, ContractType, Period	ベーススデータを取得します。

プライベート API エンドポイント

サーバーによる認証を取得するために APIKey と APISecret が必要です。

アカウントおよびトレードエンドポイント

メソッド	パラメータ	説明
ChangePositionMode	DualPosition	すべてのシンボルのユーザーのポジションモード（ヘッジモードまたはワンウェイモード）を変更します。
GetCurrentPositionMode		すべてのシンボルでユーザーのポジションモード（ヘッジモードまたは一方向モード）を取得します
NewOrder	Symbol、Side、PositionSide、Type	新しい注文を送信します。
PlaceMarketOrder	Side, Symbol, Quantity	
PlaceLimitOrder	Side、Symbol、Quantity、LimitPrice	
PlaceStopOrder	Side, Symbol, Quantity, StopPrice, LimitPrice	
PlaceTrailingStopOrder	Side, Symbol, Quantity, aActivationPrice, aCallbackRate	
QueryOrder	銘柄	注文のステータスを確認します。
CancelOrder	銘柄	アクティブな注文をキャンセルします。OrderId または OrigClientOrderId のどちらかを送信する必要があります。
CancelAllOpenOrders	銘柄	

AutoCancelAllOpenOrders	Symbol, CountdownTimer	指定されたカウントダウン終了時に指定シンボルのすべてのオープン注文をキャンセルします。
QueryCurrentOpenOrder	銘柄	
GetOpenOrders	銘柄	シンボルのすべてのオープン注文を取得します。シンボルなしでアクセスする場合は注意してください。
GetAllOrders	銘柄	すべてのアカウント注文（アクティブ、キャンセル済み、または約定済み）を取得します。
GetAccountBalance		
GetAccountInformation		現在のアカウント情報を取得します。
ChangeInitialLeverage	Symbol, Leverage	特定のシンボルマーケットのユーザーの初期レバレッジを変更します。
ChangeMarginType	Symbol, MarginType	
ModifyIsolatedPositionMargin	Symbol, Amount, Type	
GetPositionMarginChangeHistory	銘柄	
GetPositionInformation	銘柄	
GetAccountTradeList	銘柄	
GetIncomeHistory	銘柄	
GetNotionalLeverageBracket	銘柄	
TestNewOrder	Symbol, Side, PositionSide, Type	新しい注文の作成と署名/recvWindowの長さをテストします。新しい注文を作成して検証しますが、マッチングエンジンには送信しません。
ModifyOrder	銘柄	既存の注文を変更します。
NewBatchOrders	BatchOrders	複数の注文を発注します。
ModifyBatchOrders	BatchOrders	複数の注文を変更します。
CancelBatchOrders	銘柄	複数の注文をキャンセルします。
GetOrderAmendment	銘柄	注文変更履歴を取得します。
CountdownCancelAll	Symbol, CountdownTime	指定されたカウントダウン終了時に指定シンボルのすべてのオープン注文をキャンセルします。
GetForceOrders	銘柄	ユーザーの強制清算注文を取得します。
GetADLQuantile	銘柄	ポジションのADL分位数推定を取得します。
GetAccountBalanceV3		先物アカウント残高を取得します（V3）。
GetAccountInformationV3		現在のアカウント情報を取得します（V3）。
GetPositionInformationV3	銘柄	現在のポジション情報を取得します（V3）。
GetCommissionRate	銘柄	ユーザー手数料率を取得します。
GetAccountConfig		現在のアカウント設定を取得します。
GetSymbolConfig	銘柄	シンボル設定を取得します。
GetOrderRateLimit		ユーザーの注文レート制限を取得します。
GetApiTradingStatus	銘柄	API取引の定量的ルールインジケータを取得します。
ChangeMultiAssetsMode	MultiAssetsMargin	ユーザーのmulti-assetsモードを変更します。Multi-Assets Mode: true、Single-Asset Mode: false。
GetMultiAssetsMode		ユーザーの現在のマルチアセットモードを取得します。

SetFeeBurn	FeeBurn	ユーザーのBNB手数料バーンステータスを変更します。
GetFeeBurn		ユーザーの BNB 手数料バーン ステータスを取得します。
CreateListenKey		新しいユーザーデータストリームを開始します。キープアライブが送信されない場合、ストリームは 60 分後に閉じられます。
KeepAliveListenKey		タイムアウトを防ぐためにユーザーデータストリームをkeepaliveします。
CloseListenKey		ユーザーデータストリームを閉じます。

イベント

Binance Futures メッセージは TsgcWebSocketClient コンポーネントで受信されます。以下のイベントを使用できます:

OnConnect

Binance サーバーへの接続が成功した後。

OnDisconnect

Binance サーバーからの切断後

OnMessage

サーバーからクライアントに送信されたメッセージは、このイベントで処理されます。

OnError

プロトコルにエラーがある場合、このイベントが呼び出されます。

OnException

未処理の例外がある場合、このイベントが呼び出されます。

さらに、Binance APIコンポーネントには **OnBinanceHTTPException** という特定のイベントがあります。これは HTTPリクエスト (REST APIまたはWebSocketユーザーストリーム) の呼び出しにエラーが発生するたびに発生します。

(*) Binance サーバーの変更により、Rad Studio 10.1 より前の Indy バージョンではテストサーバーに接続できません。この問題は Enterprise Edition または Indy バージョンを最新にアップグレードした場合には影響しません。

API Binance Futures | 取引

Binance では REST API を使用して先物取引ができます。

設定

まず、Binance アカウントで **APIキー** を作成し、先物取引の権限を追加する必要があります。

これが完了すると、先物取引を開始できます。

まず **USDT** または **COIN** 先物で取引するかを選択する必要があります。取引したい先物コントラクトを設定できる FuturesContracts というプロパティがあります。

次に、Binance Futures Client コンポーネントで **ApiKey** と **ApiSecret** を設定します。これは、Binance サーバーへ送信される要求の署名に使用されます。

注文を発注する

新しい注文を発注するには、Binance Futures クライアントコンポーネントのメソッド **REST_API.NewOrder** を呼び出してください。

注文のタイプ(market、limitなど)に応じて、APIは多かれ少なかれ フィールドを必要とします。

Mandatory Fields

- **Symbol:** 製品IDシンボル、例: BTCUSD_210326
- **Side:** BUY または SELL
- **type:** 注文タイプ
 - LIMIT
 - MARKET
 - STOP
 - TAKE_PROFIT
 - STOP_MARKET
 - TAKE_PROFIT_MARKET
 - TRAILING_STOP_MARKET

タイプに基づく追加の必須フィールド

- **LIMIT:** timeInForce, quantity, price
- **MARKET:** quantity
- **STOP/TAKE_PROFIT:** 数量、価格、ストップ価格
- **STOP_MARKET/TAKE_PROFIT_MARKET:** stopPrice
- **TRAILING_STOP_MARKET:** callbackRate

注文を送信する際、2つの可能性があります。

1. **成功:** NewOrder 関数は Binance サーバーから送信されたメッセージを返します。
2. **エラー:** 例外は OnBinanceHttpException イベントで返されます。

API SocketIO

SocketIO

Socket.IOは、リアルタイムWebアプリケーション用のJavaScriptライブラリです。Webクライアントとサーバー間でリアルタイムの双方向通信を可能にします。2つの部分から構成されます: ブラウザーで実行されるクライアント側ライブラリと、Node.js用のサーバー側ライブラリです。両方のコンポーネントはほぼ同一のAPIを持っています。Node.jsと同様に、イベント駆動型です。

メッセージの種類

0: open (新しいトランスポートが開かれたときにサーバーから送信されます (再確認))

1: クローズ (このトランスポートのクローズをリクエストしますが、接続自体はシャットダウンしません。)

2: ping (クライアントが送信します。サーバーは同じデータを含む pong パケットで応答する必要があります。)

使用例

クライアントが送信: 2probe

server sends: 3probe

3: pong (サーバーが ping パケットに応答するために送信します。)

4: 文字列メッセージ (実際のメッセージ。クライアントとサーバーはデータとともにコールバックを呼び出す必要があります)

example:

42/chat,["join",{"room:1}"]

4はengine.ioプロトコルのメッセージパケットタイプです。

2はsocket.ioプロトコルのEVENTタイプです。

/chatはsocket.ioで処理されるデータです

socket.ioは"join"イベントを発火します。

は「room: 1」データを渡します。名前空間を省略できるのは、それが/の場合のみです。

5: アップグレード (engine.ioがトランスポートを切り替える前に、サーバーとクライアントがそのトランスポートで通信できるかどうかをテストします。このテストが成功すると、クライアントはアップグレードパケットを送信し、サーバーに古いトランスポートのキャッシュをフラッシュして新しいトランスポートに切り替えるよう要求します。)

6: noop (ノーブパケット。主にWebSocket接続が受信されたときにポーリングサイクルを強制的に発生させるために使用されます。)

プロパティ

API: SocketIOのバージョンを指定します:

ioAPI0: socket.io 0.* サーバーをサポートします (デフォルトで選択)。

ioAPI1: socket.io 1.* サーバーをサポートします

ioAPI2: socket.io 2.* サーバーをサポートします

ioAPI3: socket.io 3.* サーバーをサポートします。

ioAPI4: socket.io 4.* サーバーをサポートします

Base64: 有効にすると、バイナリメッセージはbase64として受信されます。

HandShakeCustomURL: socket.io セッションを取得するための URL のカスタマイズを可能にします。

HandShakeTimestamp: 新しいセッションがリクエストされるときにパラメータとしてタイムスタンプを送信する場合のみ有効にします (gevent-socketio Pythonサーバーにアクセスしようとする場合はこのプロパティを有効にしてください)。

HandShakeAuthToken: サーバーが認証にトークンを要求する場合は、ここに認証トークンを設定します。

Namespace: サーバーに接続するときに名前空間を設定できます。

Polling: このプロパティを無効にすると、クライアントはトランスポートとして WebSocket を使用してサーバーに直接接続します。

パラメータ: 接続パラメータを設定できます。

EncodeParameters: 有効にすると、パラメータがエンコードされます。

メソッド

WriteDataメソッドを使用して、socket.ioサーバーにメッセージを送信します (Message Typesセクションに従います)。

1. ユーザー名として John を使用して、1 つのパラメータで "add user" メソッドを呼び出します。

```
WriteData("42[\"add user\", \"John\"]");
```

イベント

OnHTTPRequest

新しい WebSocket 接続が確立される前に、socket.io サーバーはクライアントに新しいセッション id を取得するための新しい HTTP 接続を開くよう要求します。場合によっては、socket.io サーバーが HTTP ヘッダーを使用した認証を必要とします。このイベントを使用して、Basic 認証や Bearer トークン認証などのカスタム HTTP ヘッダーを追加できます。

OnAfterConnect

このイベントは socket.io 接続が成功した後に呼び出され、クライアントはサーバーにメッセージを送信できます。ここで例えばネームスペースにサブスクライブできます。

OnHTTPConnectionSSL

WebSocket サーバーがセキュアな接続を要求する場合、クライアントがサーバーに接続しようとするときこのようなエラーメッセージが表示されることがあります。

Error connecting with SSL. error:XXXXXXXX:SSL routines:ssl3_read_bytes:tlsv1 alert protocol version

このエラーは、クライアントがサーバーでサポートされていない TLS バージョンを使用して接続しようとしていることを意味します。

このエラーを解決するには、WebSocket クライアントコンポーネントの OnSSLAfterCreateHandler を処理して、より新しい TLS バージョンを設定する必要があります。

例: ここでは TLS 1.2 をプロトコルバージョンとして設定します。

```
void OnHTTPConnectionSSL(TObject *Sender; TIdSSLIOHandlerSocketBase *aSSLHandler)
{
    static_cast<tidssliohandlersocketopenssl*>(aSSLHandler)->SSLOptions->Method = sslvTLSv1_2;
}
```

API Coinbase

Coinbase

サポートされている API

- [WebSockets API](#): パブリックWebSocketサーバーに接続し、リアルタイムのマーケットデータ更新を提供します。
- [REST API](#): REST API にはアカウントと注文管理、およびパブリックマーケットデータのエンドポイントがあります。

最も一般的な使用方法

- **WebSockets API**
 - [WebSocket API への接続方法](#)
 - [WebSocket チャンネルの購読方法](#)
- **REST API**
 - [市場データの取得方法](#)
 - [Private REST APIの使用法](#)
 - [注文の発注方法](#)
 - [SandBoxアカウントの使用法](#)
 - [Private Requests Time](#)

WebSockets API

WebSocketフィードは公開されており、注文と取引のリアルタイムマーケットデータ更新を提供します。本番環境では2つのエンドポイントがサポートされています。

- **Market Data** は、注文と取引の両方の更新を提供するフィードです。ほとんどのチャンネルは認証なしで利用できるようになりました。
- **ユーザー注文データ**はユーザーの注文の更新を提供します。

以下のチャンネルをサブスクライブできます：

メソッド	Arguments	説明
SubscribeHeartBeat		すべての接続を開いたままにするためのリアルタイムサーバーPing
SubscribeStatus	aProductId : 製品の ID	すべての製品と通貨をプリセット間隔で送信します。
SubscribeCandles	aProductId : 製品の ID	製品ローソク足のリアルタイム更新
SubscribeTicker	aProductId : 製品の ID	マッチが発生するたびにリアルタイムで価格が更新されます
SubscribeTickerBatch	aProductId : 製品の ID	5000 ミリ秒ごとのリアルタイム価格更新
SubscribeLevel2	aProductId : 製品の ID	オーダーブックスナップショットを維持するすべての更新と最も簡単な方法
SubscribeMarketTrades	aProductId : 製品の ID	市場での取引が発生するたびにリアルタイム更新
SubscribeUser	aProductId : 製品の ID	認証済みユーザーを含むメッセージのみを送信します
SubscribeFuturesBalanceSummary		ユーザーの先物残高が変わるたびにリアルタイム更新

User および FuturesBalanceSummary チャンネルには認証が必要です。まず Coinbase アカウントで API キーをリクエストし、コンポーネントのプロパティ Coinbase に値を設定してください。

- ApiKey
- ApiSecret

認証を行うことで、いくつかのメリットが得られます。

1. あなたが当事者の 1 つであるメッセージは展開されており、より有用なフィールドを持っています
2. 発注したストップ注文のライフサイクル情報など、プライベートメッセージを受信します。

REST API

Private Endpoints

プライベートエンドポイントは、注文管理とアカウント管理に利用できます。

リクエストに署名する前に、Coinbase Pro の Web サイトで API キーを作成する必要があります。API キーは特定のプロファイルにスコープされます。キーの作成時に、以下の 3 つの情報を記録しておく必要があります：

- キー
- シークレット
- パスフレーズ

キーとシークレットは Coinbase Pro によってランダムに生成および提供されます。パスフレーズは API アクセスをさらに保護するためにあなたが提供します。Coinbase Pro はパスフレーズのソルト付きハッシュを検証用に保存しますが、パスフレーズを忘れた場合は回復できません。

API キーの機能を制限できます。キーを作成する前に、キーに付与する権限を選択する必要があります。権限は次のとおりです：

- View — キーに読み取りパーミッションを付与します。これにはすべての GET エンドポイントが含まれます。
- Transfer - アカウントに代わって入金・出金を含む通貨の転送を許可します。注意して有効にしてください — APIキーによる転送は二段階認証をバイパスします。
- 取引 - キーが注文を入力し、取引データを取得できるようにします。これには POST /orders といくつかの GET エンドポイントが含まれます。

Accounts

メソッド	Arguments	説明
ListAccounts		API キーのプロフィールから取引アカウントのリストを取得します。
GetAccount	aAccountId: アカウントの ID	単一アカウントの情報。account_id が分かっている場合にこのエンドポイントを使用します。API キーはアカウントと同じプロファイルに属している必要があります。

注文

メソッド	Arguments	説明
PlaceNewOrder	aOrder: 注文のすべての可能なフィールドを含むクラス	新しい注文を発注します。高度な注文オプションにアクセスする必要がある場合にのみ使用してください。
PlaceMarketOrder	aSide: buy または sell aProductId: プロダクトの ID	新しい成行注文を発注します。

	aQuoteSize: 取引ペアの2番目の資産の数量。 aBaseSize: 取引ペアの最初の資産の数量です aClient_oid: 注文を識別するために選択した注文 ID。	
PlaceLimitOrder	aSide: buy または sell aProductId: プロダクトの ID aQuoteSize: 取引ペアの第2アセットの数量。 aBaseSize: 取引ペアの最初の資産の数量 aLimitPrice: 価格制限 Client_oid: 注文を識別するために選択した注文 ID	新しいリミット注文を発注します。
PlaceStopOrder	aSide: buy または sell ProductId: 製品の ID aBaseSize: 取引ペアの最初の資産の数量 StopPrice: ストップ注文の価格 aLimitPrice: 価格の上限 aStopDirection: loss または entry Client_oid: 注文を識別するためにあなたが選択した注文ID	新しいストップオーダーを発注します
CancelOrder	aOrderId: 注文の id	以前に発注した注文をキャンセルします。注文は API キーが属するプロファイルに属している必要があります。
EditOrder	aOrderId: 注文の id aPrice: 価格 aSize: 数量	指定した新しいサイズまたは新しい価格で注文を編集します
EditOrderPreview	aOrderId: 注文の id aPrice: 価格 aSize: 数量	指定した新しいサイズまたは新しい価格で注文変更リクエストをプレビューします。
ListOrders		オプションのクエリパラメータ (product_id、order_status など) でフィルタリングされた注文のリストを取得します。
GetOrder	aOrderId: 注文の id	注文 ID で単一の注文を取得します。
PreviewOrder		注文をプレビューします。
ClosePosition	aOrderId: 注文の id aProductId: 製品の ID aSize: 数量	指定された product_id に対するすべてのオープンポジションをクローズする注文を発注します。

マーケットデータ

メソッド	Arguments	説明
GetPublicProducts		取引に利用可能な通貨ペアの一覧を取得します。
GetPublicProduct	aProductId: 製品の ID	製品IDで単一の製品の情報を取得します。
GetPublicProductBook	aProductId: 製品の ID	単一製品の板情報 (bid/ask) を取得します。limitパラメータで表示する詳細の量をカスタマイズできます。
GetPublicProductCandles	aProductId: 製品の ID aStart: 時間間隔の開始 aEnd: 時間間隔の終了です	バケット内でグループ化された単一プロダクトの製品 ID ごとのレートを取得します。

コンポーネント

	aGranularity: 各ローソク足が表す時間枠。	
GetTrades	aProductId: 製品の ID	製品IDごとに最新の取引（ティック）と最良の買値/売値のスナップショット情報を取得します。
GetTime		Coinbase Advanced API から現在の時刻を取得します。

約定

メソッド	Arguments	説明
GetFillsByOrderId		注文IDでフィルタリングされた約定リストを取得します。
GetFillsByProductId		製品 ID でフィルタリングされたフィルのリストを取得します。
GetFillsByTradeId		取引 ID でフィルタリングされたフィルのリストを取得します

変換

メソッド	Arguments	説明
CreateConvertQuote		通貨間の変換クォートを作成します。
CommitConvertTrade		convert取引をコミットします。
GetConvertTrade		コンバートトレードの詳細を取得します。

手数料

メソッド	Arguments	説明
GetTransactionSummary		取引手数料のサマリーを取得します。

製品（認証済み）

メソッド	Arguments	説明
ListProducts		利用可能な製品を一覧表示します。
GetProduct	aProductId: 製品の ID	特定の製品を取得します。
GetProductBook	aProductId: 製品の ID	製品のオーダーブックを取得します。
GetProductCandles	aProductId: 製品の ID	製品の OHLCV ローソク足を取得します。
GetMarketTrades	aProductId: 製品の ID	最近の市場取引を取得します。
GetBestBidAsk		最良の買い/売り価格を取得します。

ポートフォリオ

メソッド	Arguments	説明
ListPortfolios		すべてのポートフォリオを一覧表示します。
CreatePortfolio		新しいポートフォリオを作成します。
DeletePortfolio		ポートフォリオを削除します。
GetPortfolioBreakdown		ポートフォリオの内訳の詳細を取得します。
MovePortfolioFunds		ポートフォリオ間で資金を移動します。

Perpetuals

メソッド	Arguments	説明

GetPerpetualsPortfolioSummary		先物ポートフォリオサマリーを取得します。
ListPerpetualsPositions		パーペチュアルポジションを一覧表示します。
GetPerpetualsPosition		特定の永久ポジションを取得します。

Coinbase | WebSocket API への接続

Coinbase WebSocket APIに接続するには、新しいCoinbase APIクライアントを作成してTsgcWebSocketClientにアタッチするだけです。以下の例を参照してください:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase();
oCoinbase->Client = oClient;
oClient->Active = true;
```

Coinbase | WebSocket チャンネルのサブスクライブ

Coinbase は市場データ、注文などのリアルタイム更新を取得するためにサブスクライブできるさまざまなチャンネルを提供しています。以下に Ticker へのサブスクライブ方法のサンプルを示します:

```
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 | マーケットデータ取得

Coinbaseは、REST Endpointsを通じて公開市場データを提供します。これらのエンドポイントのいずれかを呼び出すと、リクエストした市場データのスナップショットが取得されます。
マーケットデータエンドポイントは認証を必要としないため、すべてのユーザーが自由に利用できます。

使用例: BTC-USD の注文書のスナップショットを取得するには、次の呼び出しを行います。

```
TsgcWSAPI_Coinbase oCoinbase = new TsgcWSAPI_Coinbase(this);  
ShowMessage(oCoinbase->REST_API->GetPublicProductBook("BTC-USD"));
```

Coinbase Pro | プライベート REST API

Coinbase REST API はパブリックエンドポイントとプライベートエンドポイントを提供しています。プライベートエンドポイントでは、トランザクションのセキュリティを高めるためにメッセージに署名が必要です。まず、Coinbaseアカウントにログインして新しいAPIを作成すると、以下の値を取得できます:

- ApiKey
- ApiSecret

これらのフィールドは、Coinbase APIクライアントコンポーネントのCoinbaseプロパティで設定する必要があります。

設定後、Coinbase REST API へのプライベートリクエストを開始できます。

```
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 | プライベートリクエストの時刻

Coinbaseにプライベートリクエストを行う場合、リクエストのセキュリティを高めるためにメッセージが署名されます。メッセージはローカル時刻を取得し、署名されたメッセージ内に送信します。ローカル時刻がCoinbaseサーバーと30秒を超える差がある場合、リクエストは拒否されます。そのため、ローカル時刻が同期されていることを確認することが重要です。これはOSの時刻同期メソッドを使用して行うことができます。

GetTime メソッドを呼び出して Coinbase Pro サーバーの時刻を確認できます。Coinbase Pro サーバーの時刻が返されます。

Coinbase Pro | 注文の発注

Coinbase で新しい注文を発注するには、まず非公開データにアクセスするための API キーが必要です。次の記事を確認してください: [プライベート REST API の使用方法](#)。

APIキーを設定したら、注文の発注を開始できます

Market Order

新しい成行注文を発注し、BTC-USD の 0.002 枚を購入します。

```
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
));
```

指値注文

新しい指値注文を発注し、BTC-USD の 0.002 契約を10000の指値で購入します

```
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 サンドボックスアカウント

Coinbase では、実際の資金を使わずに取引できるサンドボックスアカウントを使用できます。このアカウントには、本番アカウントとは異なる API キーの作成が必要です。
SandBox アカウントを使用するには、REST API にリクエストを行う前に **Coinbase.SandBox** プロパティを **true** に設定してください。

```
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 SignalR は、アプリにリアルタイム Web 機能を追加することを簡略化するオープンソースライブラリです。リアルタイム Web 機能により、サーバーサイドのコードがコンテンツをクライアントに即座にプッシュできます。

SignalRの適切な候補:

- サーバーからの高頻度更新を必要とするアプリ。例としてはゲーム、ソーシャルネットワーク、投票、オークション、マップ、GPS アプリなどがあります。
- ダッシュボードおよびモニタリングアプリ。例として、企業ダッシュボード、リアルタイムの売上更新、または旅行アラートが挙げられます。
- コラボレーティブアプリ。ホワイトボードアプリやチームミーティングソフトウェアはコラボレーティブアプリの例です。
- 通知を必要とするアプリ。ソーシャルネットワーク、メール、チャット、ゲーム、旅行アラート、その他多くのアプリが通知を使用します。

SignalRCore sgcWebSockets コンポーネントは、SignalRCoreサーバーに接続するためのトランスポートとして WebSocket を使用します。このトランスポートがサポートされていない場合、エラーが発生します。

Hubs

SignalRCoreは、クライアントとサーバー間の通信にハブを使用します。SignalRCoreは2つのハブプロトコルを提供します: JSON ベースのテキストプロトコルと、MessagePack ベースのバイナリプロトコルです。sgcWebSockets コンポーネントは、SignalRCoreサーバーとの通信にJSONテキストプロトコルのみを実装しています。

クライアントが使用する Hub を設定するには、クライアントがサーバーに接続する前に **SignalRCore/Hub** プロパティに Hub の名前を設定してください。

接続

クライアントがサーバーへの新しい接続を開くと、フォーマットプロトコルとバージョンを含むリクエストメッセージを送信します。sgcWebSockets は常にフォーマットプロトコルとして JSON を送信します。サーバーはプロトコルがサポートされていない場合はエラーで返信します。このエラーは **OnSignalRCoreError** イベントで処理でき、接続が成功した場合は **OnSignalRCoreConnect** イベントが呼び出されます。

クライアントがSignalRCoreサーバーに接続する際、セッション間でクライアントを識別するConnectionIdを送信できます。切断が発生した場合、クライアントは同じConnectionIdを渡してサーバーに再接続できます。新しいConnectionIdを取得するには、通常通りサーバーに接続し、**OnBeforeConnectEvent** を使用してConnectionIdを確認できます。以前のConnectionIdを渡してサーバーに再接続したい場合は、**ReConnect** メソッドを使用してパラメータとして **ConnectionId** を渡してください。

SignalRCore プロトコル

SignalR プロトコルは、任意のメッセージベーストランスポートを介した双方向 RPC のプロトコルです。接続の両当事者は相手側のプロシージャを呼び出すことができ、プロシージャは 0 個以上の結果またはエラーを返すことができます。例: クライアントはサーバーからメソッドをリクエストでき、サーバーはクライアントからメソッドをリクエストできます。サーバーとクライアント間では以下のメッセージが交換されます:

- **HandshakeRequest**: クライアントがサーバーにメッセージフォーマットに合意するために送信します。
- **HandshakeResponse**: サーバーは以前の HandshakeRequest メッセージの確認応答をクライアントに返します。ハンドシェイクが失敗した場合はエラーが含まれます。

- **Close:** 接続が閉じられたときにクライアントまたはサーバーによって呼び出されます。エラーにより接続が閉じられた場合はエラーを含みます。
- **呼び出し:** クライアントまたはサーバーが別のピアにメッセージを送信し、引数ありまたはなしでメソッドを呼び出します。
- **StreamInvocation:** クライアントまたはサーバーが、引数を伴うまたは伴わないストリーミングメソッドを呼び出すために別のピアにメッセージを送信します。レスポンスは異なるアイテムに分割されます。
- **StreamItem:** 以前の StreamInvocation からの応答です。
- **Completion:** 以前の呼び出しまたは StreamInvocation が完了したことを意味します。プロセスが成功した場合は結果を含み、エラーがある場合はエラーを含みます。
- **CancelInvocation:** 以前のStreamInvocationリクエストをキャンセルします。
- **Ping:** 接続がまだ生きているかどうかを確認するためのメッセージです。

SignalRCore エンコーディング

SignalRCore では以下のエンコーディングが使用できます:

- **JSON:** 現在サポートされている唯一のエンコーディング。
- **MessagePack**

現在、JSON のみがサポートされていますが、外部の MessagePack ライブラリを使用してメッセージをエンコードすることで MessagePack を使用できます。詳細については、以下の MessagePack セクションを参照してください。

エンコーディングプロトコルの設定は SignalRCore.Protocol プロパティで定義されます。デフォルト値は **srcpJSON** です。

認証

認証を有効にして、各接続にユーザーを関連付け、どのユーザーがリソースにアクセスできるかをフィルタリングできます。認証はBearer Tokenを使用して実装されます。クライアントはアクセストークンを提供し、サーバーはこのトークンを検証してユーザーを識別するために使用します。

標準的なWeb APIでは、ベアラートークンはHTTPヘッダーで送信されますが、WebSocketを使用する場合、トークンはクエリ文字列パラメータとして送信されます。

以下のメソッドがサポートされています:

srcaRequestToken

認証が有効な場合、フローは次のとおりです:

1. まず、サーバーから有効なトークンを取得しようとします。Authentication.RequestToken.URLに対してHTTP接続を開き、UserとPasswordのデータを使用してPOSTを行います。
2. 前のステップが成功した場合、トークンが返されます。そうでない場合はエラーが返されます。
3. トークンが返された場合、新しい接続をネゴシエートするために新しい HTTP 接続を開きます。ここでトークンは HTTP ヘッダーとして渡されます。
4. 前のステップが成功した場合、WebSocket 接続を開いてトークンをクエリ文字列パラメーターとして渡します。

- **Authentication.Enabled:** 有効な場合、WebSocket 接続が確立される前に認可が使用されます。
- **Authentication.Username:** 認証のためにサーバーに提供されるユーザー名。
- **Authentication.Password:** 認証のためにサーバーに提供するシークレットワード。
- **Authentication.RequestToken.PostFieldUsername:** ユーザー名を送信するフィールドの名前（設定によって異なります。使用される名前については HTTP JavaScript ページを確認してください）。
- **Authentication.RequestToken.PostFieldPassword:** パスワードを送信するフィールドの名前（構成によって異なります。どの名前が使用されているかを確認するにはhttp javascriptページを確認してください）。
- **Authentication.RequestToken.URL:** トークンがリクエストされる URL。
- **Authentication.RequestToken.QueryFieldToken:** websocket接続で使用するクエリ文字列パラメータの名前です。

srcasetToken

ここでは、別のサーバーから取得したトークンを SignalRCore サーバーに直接渡します。

- **Authentication.Enabled** : アクティブな場合、WebSocket 接続が確立される前に認証が使用されます。
- **Authentication.SetToken.Token** : 取得したトークンの値。

アクセストークンはクエリパラメータとして送信できます（デフォルトのオプション）。または Bearer トークンとして HTTP ヘッダーで送信することもできます。Authentication.TokenParam プロパティを使用してこの動作を設定します。

- **srctQuery** : access_token は WebSocket 接続のクエリ URL で渡されます。
- **srctHeader** : access_token は HTTP ヘッダーでベアラートークンとして渡されます。

srcaBasic

このオプションは **Basic 認証** を使用します。この認証メソッドでは SignalRCore コンポーネントと [TsgcWebSocketClient](#) の設定が必要です。

使用例: サーバーが Basic 認証を必要とし、ユーザー名が "user"、パスワードが "secret" の場合、以下に示すようにコンポーネントを設定してください。

```
// 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;
```

クライアントとサーバー間の通信

サーバーとクライアント間には3種類のやり取りがあります:

呼び出し

呼び出し元は呼び出し先にメッセージを送信し、呼び出しが完了したことを示すメッセージと、オプションで呼び出しの結果を期待します。

使用例: クライアントが SendMessage メソッドを呼び出し、ユーザー名とテキストメッセージをパラメータとして渡します。サーバーから結果メッセージを取得するために呼び出し ID を送信します。

```
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!");
    }
}
```

ノンブロッキング呼び出し

Caller は Callee にメッセージを送信し、この呼び出しに対してそれ以上のメッセージを期待しません。呼び出しは呼び出し ID 値なしで送信できます。これは呼び出しが「ノンブロッキング」であることを示します。

例: クライアントが `SendMessage` メソッドを呼び出し、パラメーターとしてユーザー名とテキストメッセージを渡します。クライアントは呼び出しの結果についてサーバーからのレスポンスを期待しません。

```
SignalRCore->Invoke("SendMessage", ARRAYOFCONST(("John", "Hello All.")));
```

ストリーミング呼び出し

Caller は Callee にメッセージを送信し、Callee から 1 つ以上の結果が返されることを期待し、その後に呼び出しの終了を示すメッセージが返されます。

例: クライアントが `Counter` メソッドを呼び出し、500 ミリ秒間隔で 10 個の数値をリクエストします。

```
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!");
    }
}
```

呼び出し

単一の呼び出しを実行するために、Caller は次の基本フローに従います:

```
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)
```

呼び出しを表す一意の Invocation ID 値（任意の文字列、Caller が選択）を割り当てます。呼び出されるターゲット、Arguments、InvocationId を含む `Invoke` または `InvokeStream` メソッドを呼び出してください（InvocationId を送信しない場合、完了結果を取得できません）。

呼び出しがノンブロッキング（下記「ノンブロッキング呼び出し」参照）とマークされている場合は、ここで停止してすぐにアプリケーションに制御を返してください。一致する呼び出し ID を持つ `StreamItem` または `Completion` メッセージを処理してください。

```
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!");
    }
}
```

単一の呼び出しを実行して完了を待つことができます。

```
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);
```

呼び出しを表す一意の呼び出し ID 値（任意の文字列、呼び出し元が選択）を割り当てます。呼び出されるターゲット、引数、および InvocationId を含む InvokeAndWait または InvokeStreamAndWait メソッドを呼び出します。プログラムは完了イベントが呼び出されるかタイムアウトが超過するまで待機します。

```
{
    TSignalRCore_Completion oCompletion;
    if (SignalRCore->InvokeStreamAndWait("Counter", ARRAYOFCONST((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);
    }
}
```

呼び出しのキャンセル

クライアントがサーバーが Completion メッセージを送信する前に StreamItem メッセージの受信を停止したい場合、クライアントはストリームを開始した StreamInvocation メッセージで使用されたのと同じ InvocationId を持つ CancelInvocation メッセージを送信できます。

```
void OnSignalRCoreStreamItem(TObject *Sender, TSignalRCore_StreamItem *StreamItem, bool &Cancel)
{
    if (StreamItem->InvocationId == "id-000002")
    {
        Cancel = true;
    }
}
```

クライアント結果

完了メッセージが受信された場合にのみ、呼び出しが完了したとみなされます。クライアントがサーバーから呼び出しを受信すると、OnSignalRCoreInvocation イベントが呼び出されます。

```
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.");

```

接続を閉じる

接続が閉じられたときにクライアントから送信されます。エラーにより接続が閉じられた場合はエラー理由が含まれます。

```

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プロトコルは、基盤となるトランスポート接続がアクティブのままであることを保証するために使用される「Keep Alive」メッセージをサポートしています。これらのメッセージは次を保証するのに役立ちます：

プロキシがアイドル時（送信されるメッセージが少ないとき）に基盤となる接続を閉じないこと。基盤となる接続が正常に終了されずに切断された場合、アプリケーションにできるだけ早く通知されます。

Keep Aliveの動作は、Pingメソッドを呼び出すか、WebSocketクライアントでHeartBeatを有効にすることで実現されます。サーバーがクライアントにpingを送信すると、クライアントは自動的に応答を送信し、OnSignalRCoreKeepAliveイベントが呼び出されます。

```

void OnSignalRCoreKeepAlive(TObject *Sender)
{
    DoLog("#keepalive");
}

```

MessagePack

SignalR プロトコルのMsgPackエンコーディングでは、各メッセージは指定されたハブプロトコルメッセージのプロパティに対応するアイテムを含む単一のMsgPack配列として表されます。配列内のアイテムはプリミティブ値、配列（例：メソッド引数）、またはオブジェクト（例：引数値）である場合があります。配列の最初のアイテムはメッセージタイプです。

送信されたメッセージのエンコード方法については [MessagePack ドキュメント](#) を参照してください。

新しいメッセージを受信するたびに、OnSignalRCoreMessagePack イベントでディスパッチされます。メッセージは Data ストリームパラメーターを読み取ることでアクセスできます。デフォルトでは JSON パラメーターは空です。MessagePack メッセージを JSON に変換すると、コンポーネントはエンコードが JSON を使用していた場合と同様に JSON メッセージを処理します（OnSignalRCoreCompletion、OnSignalRCoreInvocation などのイベントがディスパッチされます）。

API SignalR

SignalR

SignalR コンポーネントは、SignalR サーバーへの接続にトランスポートとして WebSocket を使用します。このトランスポートがサポートされていない場合、エラーが発生します。

SignalRクライアントコンポーネントには、次のデータを設定できるSignalRというプロパティがあります:

- **Hubs:** クライアントがサブスクライブするハブのリストを含みます。
- **ProtocolVersion:** クライアントが使用するプロトコルのバージョン。プロトコルバージョン 1.2 から 1.5 をサポートします。
- **UserAgent:** SignalR サーバーへの接続に使用するユーザーエージェント。

クライアントはテキストまたはバイナリデータの送信をサポートしています。

ハブメッセージ

Hubs API を使用すると、クライアントからサーバーメソッドを、サーバーからクライアントメソッドを呼び出せます。永続接続に使用されるプロトコルは、RPC（リモートプロシージャ呼び出し）セマンティクスを表現できるほどリッチではありません。ただし、これはハブ接続に使用されるプロトコルが、永続接続に使用されるプロトコルとまったく異なるという意味ではありません。むしろ、ハブ接続に使用されるプロトコルは、ほとんどが永続接続用プロトコルの拡張です。

クライアントがサーバーメソッドを呼び出すとき、永続的な接続のような自由形式の文字列は送信しなくなります。代わりに、メソッドの呼び出しに必要な情報をすべて含む JSON 文字列を送信します。クライアントがサーバーメソッドを呼び出すために送信するサンプルメッセージを以下に示します:

```
WriteData("{\"H\":\"chathub\",\"M\":\"Send\",\"A\":[\"CBuilder Client\",\"Test message\"],\"I\":0}
```

ペイロードには以下のプロパティがあります。

I — 呼び出し識別子 — レスポンスをリクエストと照合するために使用されます

H — ハブの名前

M — メソッドの名前

A — arguments(配列。メソッドにパラメータがない場合は空でも構いません)

文字列引数に**ダブルクォート**がある場合は「」を「\」に置き換えてください

例: 引数が {"test":1} の場合、引数を {"test\\":1} として送信します

```
WriteData("{\"H\":\"chathub\",\"M\":\"Send\",\"A\":[\"test\\\":1],\"I\":0});
```

認証

認証を有効にして、各接続にユーザーを関連付け、どのユーザーがリソースにアクセスできるかをフィルタリングできます。認証はBearer Tokenを使用して実装されます。クライアントはアクセストークンを提供し、サーバーはこのトークンを検証してユーザーを識別するために使用します。

現在 Bearer トークンのみがサポートされています:

ここでは、トークンをSignalサーバーへ直接渡します（トークンは別のサーバーから取得済みのためです）。

- **Authentication.Enabled:** アクティブな場合、WebSocket 接続が確立される前に認証が使用されます。

- **Authentication.Authentication:** 2 種類の認証がサポートされています: Bearer トークンまたは Cookie。どちらも必要な値を取得する外部の方法が必要です。
 - **BearerToken:** 取得されたトークンの値。
 - **Cookie:** 必要な Cookie の値を設定します。

```
TsgcWSAPI_Signal *oSignalR = new TsgcWSAPI_Signal(NULL);
oSignalR->SignalR->Enabled = true;
oSignalR->SignalR->Authentication = srcBearerToken;
oSignalR->SignalR->BearerToken->Token = "token here";
```

コンポーネントには次のイベントがあります:

OnSignalRConnect

このイベントはクライアントがサーバーに正常に接続したときに発生します。

OnSignalRDisconnect

このイベントはクライアントがサーバーから切断されたときに発生します。

OnSignalRError

このイベントは、WebSocket接続にエラーがあるときに呼び出されます。

OnSignalRMessage

永続接続に使用されるプロトコルは非常にシンプルです。サーバーに送信されるメッセージは単なる生の文字列です。特定の形式である必要はありません。クライアントに送信されるメッセージはより構造化されています。メッセージに含まれるプロパティは次のとおりです:

C — メッセージ ID、KeepAlive 以外のすべてのメッセージに存在します
M — 実際のデータを含む配列。

```
{"C": "d-9B7A6976-B, 2|C, 2", "M": ["Welcome!"]}
```

OnSignalRBinary

このイベントはサーバーからバイナリデータを受信したときに呼び出されます。

OnSignalRResult

サーバーメソッドが呼び出されると、サーバーは呼び出しが完了したことを確認するために呼び出し ID をクライアントに送信します。また、メソッドが戻り値を持つ場合は戻り値も送信し、メソッドの呼び出しに失敗した場合はエラーを送信します。

サーバーメソッド呼び出しの結果サンプルを以下に示します:

```
{"I": "0"}
```

呼び出し識別子が「0」のサーバーvoidメソッドが正常に完了しました。

```
{"I":"0", "R":42}
```

呼び出し識別子が "0" のサーバー メソッドが正常に完了し、値 42 を返しました。

```
{"I":"0", "E":"Error occurred"}
```

OnSignalRKeepAlive

サーバーから KeepAlive メッセージを受信したときに発生します。

API Kraken

Kraken

概要

WebSockets API はリアルタイムの市場データ更新を提供します。WebSockets は双方向プロトコルで、最も速いリアルタイムデータを提供し、リアルタイムアプリケーションの構築を支援します。以下に示すパブリックメッセージタイプは認証を必要としません。プライベートデータのメッセージは別の認証済みエンドポイントでサブスクライブできます。

Kraken は公開マーケットデータとプライベートユーザーデータ（認証が必要）を提供する REST API も提供しています。

設定

プライベートAPIでは、Krakenアカウントから作成したAPIキーが必要です。Kraken では WebSocket プロトコルのテスト環境が利用可能です。このベータ機能を使用するには、Kraken プロパティの Beta プロパティを有効にしてください。

サポートされている API

- [WebSockets パブリック API](#): パブリック WebSocket サーバーに接続します。
- [WebSockets プライベート API](#): プライベート WebSocket サーバーに接続し、サーバーに対して認証するために API キーと API シークレットが必要です。
- [REST パブリック API](#): パブリック REST サーバーに接続します。
- [REST プライベート API](#): パブリック REST サーバーに接続し、サーバーへの認証に API キーと API シークレットが必要です。

Kraken の例

Public WebSocket Serverへの接続方法

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient(NULL);
TsgcWSAPI_Kraken *oKraken = new TsgcWSAPI_Kraken(NULL);
oKraken->Client = oClient;
oClient->Active = true;
```

プライベート WebSocket サーバーへの接続方法

```
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 API からティッカーを取得する方法

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient(NULL);
TsgcWSAPI_Kraken *oKraken = new TsgcWSAPI_Kraken(NULL);
oKraken->Client = oClient;
ShowMessage(oKraken->GetTicker(ARRAYOFCONST(("XBTUSD"))));
```

REST API メソッド

パブリックエンドポイント

メソッド	Arguments	説明
GetSystemStatus		現在のシステムステータスを取得します。

Private Endpoints

メソッド	Arguments	説明
GetExtendedBalance		拡張された残高情報を取得します。
AmendOrder		既存の注文を修正します。
CancelAllOrders		すべての未決注文をキャンセルします。
CancelAllOrdersAfter		デッドマンズスイッチ - タイムアウト後にすべての注文をキャンセル。
EditOrder		既存の注文を編集します。
AddOrderBatch		複数の注文を一括追加します。
CancelOrderBatch		複数の注文を一括キャンセルします。
GetWithdrawalMethods		利用可能な出金方法を取得します。
GetWithdrawalAddresses		出金アドレスを取得します。

REST API からアカウント残高を取得する方法

```
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 | WebSocket パブリック API

接続

URL: `wss://ws.kraken.com (v1)` または `wss://ws.kraken.com/v2 (v2、推奨)`

このコンポーネントは **Version** プロパティ（デフォルト: 2）を通じて WebSocket API v2 をサポートするようになりました。レガシー v1 エンドポイントを使用するには `Kraken.Version := 1` を設定してください。

ソケットが開いたら、サブスクライブリクエストメッセージを送信することでパブリックチャンネルをサブスクライブできます。

一般的な考慮事項

- WebSocket を通じて送受信されるすべてのメッセージは JSON 形式でエンコードされます。
- すべての浮動小数点フィールド（タイムスタンプを含む）は、精度を保持するために引用符で囲まれます。
- 各取引可能ペアのフォーマットは A/B であり、A と B は標準化された資産については ISO 4217-A3、標準化されていない場合は一般的なユニークシンボルです。
- タイムスタンプは一意であるとはみなされず、トランザクション ID の別名とも見なされません。また、タイムスタンプの粒度はトランザクションレートを表すものではありません。

対応ペア

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

メソッド

Ping

クライアントはサーバーに ping を送信して接続が生きているかどうかを確認でき、サーバーは pong で応答します。

これはサーバーが開始する WebStandards 標準のデフォルト ping とは対照的な、アプリケーションレベルの ping です。

ティッカー

ティッカー情報には、指定した通貨ペアのベストアスク価格とベストビッド価格、24時間の取引量、最終取引価格、出来高加重平均価格などが含まれます。ティッカーメッセージは取引または取引グループが発生するたびに公開されます。

SubscribeTicker メソッドを呼び出してティッカーを購読します:

```
SubscribeTicker(["XBT/USD"]);
```

サブスクリプションが成功すると、**OnKrakenSubscribed** イベントが呼び出されます。

```
void OnKrakenSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription, string ChannelName,
int ReqID)
{
    DoLog("#subscribed: " + Subscription + " " + Pair + " " + ChannelName);
}
```

UnSubscribeTicker メソッドを呼び出してサブスクリプションを解除します:

```
UnSubscribeTicker(ARRAYOFCONST((L"XBT/USD")));
```

サブスクリプション解除が成功すると、OnKrakenUnSubscribed イベントが呼び出されます:

```
void OnKrakenUnSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
int ReqID)
{
    DoLog("#unsubscribed: " + Subscription + " " + Pair);
}
```

サブスクライブ/アンサブスクライブ中にエラーが発生した場合、OnKrakenSubscriptionError イベントが呼び出されます。

```
void OnKrakenSubscriptionError(TObject *Sender, string ErrorMessage, string Pair, string Subscription,
int ReqID)
{
    DoLog("#subscription error: " + ErrorMessage);
}
```

Ticker の更新は OnKrakenData イベントで通知されます。

```
[
    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

OHLCをサブスクライブすると、（endtimeに関係なく）最後の有効なローソク足のスナップショットが送信され、続いて進行中のローソク足の更新が送信されます。たとえば、1分足にサブスクライブして5分間約定がなかった場合、5分前の最後の1分足のスナップショットが発行されます。endtimeを使用して、それが古いローソク足であると判断できます。

SubscribeOHLC メソッドを呼び出してOHLCにサブスクライブします。ペアとインターバルを渡す必要があります。

```
SubscribeOHLC(ARRAYOFCONST((L"XBT/USD")), kin1min);
```

サブスクリプションが成功した場合、OnKrakenSubscribed イベントが呼び出されます:

```
void OnKrakenSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription, string ChannelName,
    int ReqID)
{
    DoLog("#subscribed: " + Subscription + " " + Pair + " " + ChannelName);
}
```

UnSubscribeOHLC メソッドを呼び出して登録解除する:

```
UnSubscribeOHLC(ARRAYOFCONST(("XBT/USD")), kin1min);
```

サブスクリプション解除が成功すると、OnKrakenUnSubscribed イベントが呼び出されます:

```
void OnKrakenUnSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
    int ReqID)
{
    DoLog("#unsubscribed: " + Subscription + " " + Pair);
}
```

サブスクライブ/アンサブスクライブ中にエラーが発生した場合、OnKrakenSubscriptionError イベントが呼び出されます。

```
void OnKrakenSubscriptionError(TObject *Sender, string ErrorMessage, string Pair, string Subscription,
    int ReqID)
{
    DoLog("#subscription error: " + ErrorMessage);
}
```

OHLC の更新は OnKrakenData イベントで通知されます。

```
[
    42,
    [
        "1542057314.748456",
        "1542057360.435743",
        "3586.70000",
        "3586.70000",
        "3586.60000",
        "3586.60000",
        "3586.68894",
        "0.03373000",
        2
    ],
    "ohlcv-5",
    "XBT/USD"
]
```

取引

通貨ペアのトレードフィード。

SubscribeTrade メソッドを呼び出して取引フィードをサブスクライブします。

```
SubscribeTrade(ARRAYOFCONST((L"XBT/USD")));
```

サブスクリプションが成功した場合、OnKrakenSubscribed イベントが呼び出されます:

```
void OnKrakenSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
    string ChannelName, int ReqID)
{
    DoLog("#subscribed: " + Subscription + " " + Pair + " " + ChannelName);
}
```

UnSubscribeTradeメソッドを呼び出してアンサブスクライブします:

```
UnSubscribeTrade(ARRAYOFCONST((L"XBT/USD")));
```

サブスクリプション解除が成功すると、OnKrakenUnSubscribed イベントが呼び出されます:

```
void OnKrakenUnSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
    int ReqID)
{
    DoLog("#unsubscribed: " + Subscription + " " + Pair);
}
```

サブスクライブ/アンサブスクライブ中にエラーが発生した場合、OnKrakenSubscriptionError イベントが呼び出されます。

```
void OnKrakenSubscriptionError(TObject *Sender, string ErrorMessage, string Pair, string Subscription,
    int ReqID)
{
    DoLog("#subscription error: " + ErrorMessage);
}
```

約定の更新はOnKrakenDataイベントで通知されます。

```
[
  0,
  [
    [
      "5541.20000",
      "0.15850568",
      "1534614057.321597",
      "s",
      "l",
      ""
    ],
    [
      "6060.00000",
      "0.02455000",
      "1534614057.324998",
      "b",
      "l",
      ""
    ]
  ],
  "trade",
  "XBT/USD"
]
```

ブック

注文板レベル。サブスクリプション時には指定された深さのスナップショットが発行され、その後レベルの更新が発行されます。

SubscribeBook メソッドを呼び出してブックをサブスクライブします。ペアと深度を渡す必要があります。

```
SubscribeBook(ARRAYOFCONST((L"XBT/USD")), kde10);
```

サブスクリプションが成功した場合、OnKrakenSubscribed イベントが呼び出されます:

```
void OnKrakenSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription, string ChannelName,
    int ReqID)
{
```

```
DoLog("#subscribed: " + Subscription + " " + Pair + " " + ChannelName);
}
```

UnSubscribeBook メソッドを呼び出してサブスクリプションを解除します:

```
UnSubscribeBook(ARRAYOFCNST((L"XBT/USD")), kde10);
```

サブスクリプション解除が成功すると、OnKrakenUnSubscribed イベントが呼び出されます:

```
void OnKrakenUnSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
int ReqID)
{
DoLog("#unsubscribed: " + Subscription + " " + Pair);
}
```

サブスクライブ/アンサブスクライブ中にエラーが発生した場合、OnKrakenSubscriptionError イベントが呼び出されます。

```
void OnKrakenSubscriptionError(TObject *Sender, string ErrorMessage, string Pair, string Subscription,
int ReqID)
{
DoLog("#subscription error: " + ErrorMessage);
}
```

Book更新はOnKrakenDataイベントで通知されます。

```
[
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"
]
```

スプレッド

サブスクライブした資産ペアの最良買い値と売り値を表示するスプレッドフィード。買い量と売り量もメッセージに含まれています。

SubscribeSpread メソッドを呼び出してスプレッドフィードにサブスクライブします。


```
SubscribeSpread(ARRAYOFCONST(("XBT/USD")));
```

サブスクリプションが成功した場合、OnKrakenSubscribed イベントが呼び出されます:

```
void OnKrakenSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription, string ChannelName,
int ReqID)
{
    DoLog("#subscribed: " + Subscription + " " + Pair + " " + ChannelName);
}
```

UnSubscribeSpread メソッドを呼び出してアンサブスクライブします:

```
UnSubscribeSpread(ARRAYOFCONST(("XBT/USD")));
```

サブスクリプション解除が成功すると、OnKrakenUnSubscribed イベントが呼び出されます:

```
void OnKrakenUnSubscribed(TObject *Sender, int ChannelId, string Pair, string Subscription,
int ReqID)
{
    DoLog("#unsubscribed: " + Subscription + " " + Pair);
}
```

サブスクライブ/アンサブスクライブ中にエラーが発生した場合、OnKrakenSubscriptionError イベントが呼び出されます。

```
void OnKrakenSubscriptionError(TObject *Sender, string ErrorMessage, string Pair, string Subscription,
int ReqID)
{
    DoLog("#subscription error: " + ErrorMessage);
}
```

スプレッドの更新は OnKrakenData イベントで通知されます。

```
[
    0,
    [
        "5698.40000",
        "5700.00000",
        "1542057299.545897",
        "1.01234567",
        "0.98765432"
    ],
    "spread",
    "XBT/USD"
]
```

その他のメソッド

1 つのメソッドですべてのチャンネルをサブスクライブ/アンサブスクライブできます:

```
SubscribeAll(ARRAYOFCONST(("XBT/USD")));
UnSubscribeAll(ARRAYOFCONST(("XBT/USD")));
```

すべてのチャンネルが購読されている場合、OHLC インターバル値は 1 です。

イベント

OnConnect: WebSocket クライアントがサーバーに接続されたとき。

OnKrakenConnect: WebSocket 接続の成功後およびサーバーがシステムステータスを送信したときに呼び出されます。

OnKrakenSystemStatus: システムステータスが変化したときに呼び出されます。

OnKrakenSubscribed: チャンネルへのサブスクリプションが成功した後に呼び出されます。

OnKrakenUnSubscribed: チャンネルからの購読解除が成功した後に呼び出されます。

OnKrakenSubscriptionError: 購読または購読解除の試行中にエラーが発生した場合に呼び出されます。

OnKrakenData: チャンネルサブスクリプションに更新がある度に呼び出されます。

API Kraken | WebSockets プライベート API

接続

URL: wss://ws-auth.kraken.com (v1) または wss://ws-auth.kraken.com/v2 (v2、推奨)

コンポーネントは **Version** プロパティ（デフォルト：2）を通じて WebSocket API v2 をサポートするようになりました。

ソケットが開かれると、認証済みのサブスクライプリクエストメッセージを送信してプライベートデータチャンネルをサブスクライブできます。

認証

API クライアントは WebSocket プライベートエンドポイントに接続するために REST API エンドポイント「GetWebSocketsToken」で Authentication 「token」をリクエストする必要があります。トークンは作成後15分以内に使用する必要があります。トークンは、WebSocket API プライベートメッセージ（openOrders または ownTrades）への接続が維持されている限り期限切れになりません。

WebSocket トークンを取得するには、API キーと API シークレットを アカウントで Kraken が提供する Kraken オプションコンポーネントに設定する必要があります。

```
Kraken->ApiKey = "api key";  
Kraken->ApiSecret = "api secret";
```

メソッド

OwnTrades

自分の約定の一覧を取得します。最初のサブスクリプション時には、直近の 50件の約定の一覧を取得します

```
SubscribeOwnTrades();
```

後で UnSubscribeOwnTrades メソッドを呼び出すことで OwnTrades のサブスクリプションを解除できます。

```
UnSubscribeOwnTrades();
```

サーバーからのレスポンス例

```
[  
  [  
    {  
      "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

使用された API キーに認証されたユーザーに属するすべてのオープン注文を表示するフィード。初期スナップショットはすべてのオープン注文のリストを提供し、その後オープン注文リストへの更新が送信されます。「closed」などのステータス変更の更新については、orderid と status フィールドがペイロードに含まれます。

```
SubscribeOpenOrders();
```

後でOpenOrdersのサブスクリプションを解除するには、UnSubscribeOpenOrdersメソッドを呼び出します。

```
UnSubscribeOpenOrders();
```

サーバーからのレスポンス例

```
[
  [
    {
      "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

Krakenに新しい注文を送信

```
TsgcWSKrakenOrder *oKrakenOrder = new TsgcWSKrakenOrder();
oKrakenOrder->Pair = "XBT/USD";
oKrakenOrder->_Type = kosBuy;
oKrakenOrder->OrderType = kotMarket;
oKrakenOrder->Volume = 1;
AddOrder(oKrakenOrder);
```

注文パラメータのリスト

```
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

```

サーバーからのレスポンス例

```

{
  "descr": "buy 0.01770000 XBTUSD @ limit 4000",
  "event": "addOrderStatus",
  "status": "ok",
  "txid": "ONPNXH-KMKMU-F4MR5V"
}

```

注文のキャンセル

注文をキャンセルする

```
CancelOrder("Order Id");
```

サーバーからのレスポンス例

```

{
  "event": "cancelOrderStatus",
  "status": "ok"
}

```

API Kraken | REST パブリック API

接続

URL: <https://api.kraken.com>

Kraken Public API は認証を必要としません。

設定

唯一の設定は、REST HTTP リクエストのログを有効にするかどうかです。すべての HTTP リクエスト/レスポンスをテキストファイルに保存する場合は HTTPLogOptions を有効にします。

イベント

OnKrakenHTTPException: このイベントは、REST Api からHTTP要求を行う際に例外が発生した場合に呼び出されます。

メソッド

GetServerTime

このメソッドはサーバーとクライアント間のスキュー時間を近似するために役立ちます。Unix 形式の時刻を返します。

```
{"error": [], "result": {"unixtime": 1586705546, "rfc1123": "Sun, 12 Apr 20 15:32:26 +0000"}}
```

GetAssets

アセットに関する情報を返します

```
{"error": [], "result": {"ADA": {"aclass": "currency", "altname": "ADA", "decimals": 8, "display_decimals": 6}}}}
```

GetAssetPairs

アセットのペアに関する情報を返します

```
Kraken->REST_API->GetAssetPairs(ARRAYOFCONST((L"XBTUSD")));
```

GetTicker

ティッカー情報を返します。

```
Kraken->REST_API->GetTicker(ARRAYOFCONST((L"XBTUSD")));
```

GetOHLC

Open-High-Low-Closeデータを返します。

```
Kraken->REST_API->GetOHLC("XBTUSD");
```

GetOrderBook

ペア名と板の深さの配列を返します。

```
Kraken->REST_API->GetOrderBook("XBTUSD");
```

GetTrades

ペアの最近の取引データを返します。

```
Kraken->REST_API->GetTrades("XBTUSD");
```

GetSpread

ペアの最近のスプレッドデータを返します。

```
Kraken->REST_API->GetSpread("XBTUSD");
```

API Kraken | REST プライベート API

接続

URL: <https://api.kraken.com>

認証

REST Private API には API キーと API シークレットが必要です。これらの値はアカウントの Kraken によって提供されます。

```
Kraken->ApiKey = "api key";  
Kraken->ApiSecret = "api secret";
```

メソッド

GetAccountBalance

アカウント残高を返します。

```
Kraken->REST_API->GetAccountBalance();
```

GetTradeBalance

あなたの取引に関する情報を返します。

```
Kraken->REST_API->GetTradeBalance();
```

GetOpenOrders

オープン注文のリストを返します。

```
Kraken->REST_API->GetOpenOrders();
```

GetClosedOrders

クローズされた注文のリストを返します。

```
Kraken->REST_API->GetClosedOrders();
```

QueryOrders

注文に関する情報を照会します。

```
Kraken->REST_API->QueryOrders("1234");
```


GetTradesHistory

取引情報の配列を返します。

```
Kraken->REST_API->GetTradesHistory();
```

QueryTrades

取引に関する情報を照会します。

```
Kraken->REST_API->QueryTrades("1234");
```

GetOpenPositions

ポジション情報を返します。

```
Kraken->REST_API->GetOpenPositions("1234");
```

GetLedgers

台帳情報の連想配列を返します。

```
Kraken->REST_API->GetLedgers();
```

QueryLedgers

台帳情報の連想配列を返します。

```
Kraken->REST_API->QueryLedgers("1234");
```

GetTradeVolume

取引量情報を返します。

```
Kraken->REST_API->GetTradeVolume();
```

AddExport

新しいレポートエクスポートを追加します。

```
Kraken->REST_API->AddExport("Report All Trades");
```

ExportStatus

レポートのステータスを取得する

```
Kraken->REST_API->ExportStatus();
```

RetrieveExport

レポート ID でレポートを取得します。

```
Kraken->REST_API->RetrieveExport("GOCO");
```

RemoveExport

レポート ID でレポートを削除します。

```
Kraken->REST_API->RemoveExport("GOCO");
```

Add Order

新しい注文を追加します。

```
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

ID でオープン注文をキャンセルします

```
Kraken->REST_API->CancelOrder("1234");
```

API Kraken Futures

Kraken Futures

概要

REST API を使用すると、Kraken Futures アカウントのメソッドに安全にアクセスできます。REST API メソッドの例：

- 現在または過去の価格情報をリクエストします
- アカウント残高と損益を確認します。
- マージンパラメータと推定清算 閾値
- 注文の発注またはキャンセル（個別またはバッチ）
- 未約定の注文を確認します
- オープンポジションまたは取引履歴
- デジタル資産の出金をリクエストします。

これらのメソッドは「エンドポイント」と呼ばれ、REST API セクションで説明されています。

WebSocket API を使用すると、Kraken Futures プラットフォームへのセキュアな通信チャネルを確立し、リアルタイムで情報を受信できます。これにより、継続的にリクエストを送信する代わりに更新をリッスンできます。これらのチャネルはサブスクリプションと呼ばれます。

一部のエンドポイントでは、デジタル資産の出金の開始など、機密性の高いタスクを実行できます。これらのエンドポイントに安全にアクセスするために、API は国家安全保障局が開発した暗号化技術を使用します。

設定

API を使用するには、一意の **API キー** のペアを生成する必要があります（プライベート API にアクセスする場合）：

1. **Kraken Futures アカウント** にサインインします。
2. 右上隅の名前をクリックします。
3. ドロップダウンメニューから「設定」を選択します。
4. API パネルで「Create Key」タブを選択します。
5. 「Create Key」ボタンを押します。
6. パブリックキーとプライベートキーを確認し、安全な場所に記録してください。

公開鍵と秘密鍵を、コンポーネントの **KrakenOptions** プロパティにコピーします。

```
KrakenOptions.ApiKey  
KrakenOptions.ApiSecret
```

サポートされている API

- **WebSockets 公開 API**: 公開 WebSocket サーバーに接続します。
- **WebSockets プライベート API**: プライベート WebSocket サーバーに接続し、サーバーに対して認証するための API キーと API シークレットが必要です。
- **REST Public API**: パブリック REST サーバーに接続します。
- **REST プライベート API**: パブリック REST サーバーに接続し、サーバーへの認証に API Key と API Secret が必要です。

REST API メソッド

Private Endpoints

メソッド	Arguments	説明
BatchOrder		バッチ注文の送信（発注/キャンセル/編集）。
GetOrderStatus		特定の注文のステータスを取得します。
GetPNLCurrencyPreferences		PNL 通貨設定を取得します。
SetPNLCurrencyPreference		シンボルの PNL 通貨設定を設定します。
GetLeverageSettings		レバレッジ設定を取得します。
SetLeverageSettings		シンボルのレバレッジを設定します。

API Kraken Futures | WebSockets パブリック API

接続

URL: `wss://futures.kraken.com/ws/v1`

ソケットが開いたら、サブスクライブリクエストメッセージを送信することでパブリックチャンネルをサブスクライブできます。

メソッド

ティックー

このエンドポイントは、現在上場されているすべての Futures コントラクトとインデックスの現在の市場データを返します。認証は不要です。

SubscribeTicker メソッドを呼び出してティックーをサブスクライブします。

```
SubscribeTicker(ARRAYOFCONST(("PI_XBTUSD")));
```

サブスクリプションが成功すると、**OnKrakenFuturesSubscribed** イベントが呼び出されます:

```
void OnKrakenFuturesSubscribed(TObject *Sender, string Feed, string ProductId)
{
    DoLog("#subscribed: " + Feed + " " + ProductId);
}
```

UnSubscribeTicker メソッドを呼び出してサブスクリプションを解除します。

```
UnSubscribeTicker(ARRAYOFCONST(("PI_XBTUSD")));
```

サブスクリプション解除が成功すると、**OnKrakenFuturesUnSubscribed** イベントが呼び出されます:

```
void OnKrakenFuturesUnSubscribed(TObject *Sender, string Feed, string ProductId)
{
    DoLog("#unsubscribed: " + Feed + " " + ProductId);
}
```

サブスクライブ/アンサブスクライブ中にエラーが発生した場合、**OnKrakenFuturesError** イベントが呼び出されます。

```
void OnKrakenFuturesError(TObject *Sender, string Error)
{
    DoLog("#error: " + Error);
}
```

ティックーの更新は**OnKrakenData** イベントで通知されます。

```
{    "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"

}

```

取引

トレードフィードは約定した取引に関する情報を返します。
SubscribeTradeメソッドを呼び出して、Tradeフィードをサブスクライブします。

```
SubscribeTrade(ARRAYOFCONST(("PI_XBTUSD")));
```

サブスクリプションが成功すると、**OnKrakenFuturesSubscribed** イベントが呼び出されます:

```

void OnKrakenFuturesSubscribed(TObject *Sender, string Feed, string ProductId,)
{
    DoLog("#subscribed: " + Feed + " " + ProductId);
}

```

UnSubscribeTrade メソッドを呼び出してアンサブスクライブします:

```
UnSubscribeTrade(ARRAYOFCONST(("PI_XBTUSD")));
```

サブスクリプション解除が成功すると、**OnKrakenFuturesUnSubscribed** イベントが呼び出されます:

```

void OnkrakenFuturesUnSubscribed(TObject *Sender, string Feed, string ProductId)
{
    DoLog("#unsubscribed: " + Feed + " " + ProductId);
}

```

サブスクライブ/アンサブスクライブ中にエラーが発生した場合、**OnKrakenFuturesError** イベントが呼び出されます。

```

void OnKrakenFuturesError(TObject *Sender, string Error)
{
    DoLog("#error: " + Error);
}

```

取引の更新は **OnKrakenData** イベントで通知されます。

```
{  "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

    }

  ]

}
```

ブック

このフィードはオーダーブックに関する情報を返します。
SubscribeBook メソッドを呼び出して本帳をサブスクライブします。シンボルを渡す必要があります。

```
SubscribeBook(ARRAYOFCONST(("PI_XBTUSD")));
```

サブスクリプションが成功すると、**OnKrakenFuturesSubscribed** イベントが呼び出されます:


```
void OnKrakenFuturesSubscribed(TObject *Sender, string Feed, string ProductId)
{
    DoLog("#subscribed: " + Feed + " " + ProductId);
}
```

UnSubscribeBook メソッドを呼び出して購読解除します。

```
UnSubscribeBook(ARRAYOFCONST(("PI_XBTUSD")));
```

サブスクリプション解除が成功すると、**OnKrakenFuturesUnSubscribed** イベントが呼び出されます:

```
void OnKrakenFuturesUnSubscribed(TObject *Sender, string Feed, string ProductId)
{
    DoLog("#unsubscribed: " + Feed + " " + ProductId);
}
```

サブスクライブ/アンサブスクライブ中にエラーが発生した場合、**OnKrakenFuturesError** イベントが呼び出されます。

```
void OnKrakenFuturesError(TObject *Sender, string Error)
{
    DoLog("#error: " + Error);
}
```

ブックの更新は**OnKrakenData**イベントで通知されます。

```
{
    "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

ティックーライトフィードは、上場製品のティックー情報を返します。
SubscribeTickerLite メソッドを呼び出してスプレッドフィードを購読します。

```
SubscribeTickerLite(ARRAYOFCONST(("PI_XBTUSD")));
```

サブスクリプションが成功すると、**OnKrakenFuturesSubscribed** イベントが呼び出されます:

```
void OnKrakenFuturesSubscribed(TObject *Sender, string Feed, string ProductId)
{
  DoLog("#subscribed: " + Feed + " " + ProductId);
}
```

UnSubscribeTickerLite メソッドを呼び出してUnSubscribeします:

```
UnSubscribeTickerLite(ARRAYOFCONST(("PI_XBTUSD")));
```

サブスクリプション解除が成功すると、**OnKrakenFuturesUnSubscribed** イベントが呼び出されます:

```
void OnKrakenFuturesUnSubscribed(TObject *Sender, string Feed, string ProductId)
{
  DoLog("#unsubscribed: " + Feed + " " + ProductId);
}
```

サブスクライブ/アンサブスクライブ中にエラーが発生した場合、**OnKrakenFuturesError** イベントが呼び出されます。

```
void OnKrakenFuturesError(TObject *Sender, string Error)
{
  DoLog("#error: " + Error);
}
```

スプレッドの更新は **OnKrakenData** イベントで通知されます。

```
{  "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

ハートビートフィードは定期的な間隔でハートビートメッセージを発行します。

```
SubscribeHeartBeat();  
UnSubscribeHeartBeat();
```

イベント

OnConnect: WebSocket クライアントがサーバーに接続されたとき。

OnKrakenFuturesConnect: WebSocket接続の成功後、サーバーがシステムステータスを送信したときに呼び出されます。

OnKrakenFuturesSubscribed: チャンネルへの購読が成功した後に呼び出されます。

OnKrakenFuturesUnSubscribed: チャンネルのサブスクライブ解除が成功した後に呼び出されます。

OnKrakenFuturesError: サブスクライブ/アンサブスクライブ中にエラーが発生した場合に呼び出されます。

OnKrakenData: チャンネルサブスクリプションに更新がある度に呼び出されます。

API Kraken Futures | WebSockets Private API

接続

URL: `wss://futures.kraken.com/ws/v1`

認証

WebSocketプライベートフィードへのサブスクライブおよびアンサブスクライブのリクエストには、ユーザーの `api_secret` で署名されたチャレンジメッセージが必要です。チャレンジは、WebSocket API パブリックセクション (`api_key` を使用) に示されている方法で取得します。認証済みリクエストには、元のチャレンジメッセージ (`original_challenge`) と署名済みチャレンジ (`signed_challenge`) の両方を JSON 形式で含める必要があります。Websockets チャレンジを取得するには、API キーと API シークレットを Kraken Options コンポーネントに設定する必要があります。API キーはアカウントの Kraken から提供されます。

```
Kraken->ApiKey = "api key";
Kraken->ApiSecret = "api secret";
```

メソッド

未決注文詳細

このサブスクリプションフィードは、ユーザーのオープンオーダーに関する情報を公開します。このフィードは、板に約定しなかったすべての post-only 注文に関する追加情報を提供します。

```
SubscribeOpenOrdersVerbose();
```

後で **UnSubscribeOpenOrdersVerbose** メソッドを呼び出して `OpenOrdersVerbose` からサブスクライブ解除できます。

```
UnSubscribeOpenOrdersVerbose();
```

サーバーからのレスポンス例

```
{
  'feed': 'open_orders_verbose_snapshot',
  'account': '0f9c23b8-63e2-40e4-9592-6d5aa57c12',
  {
    'instrument': 'PI_XBTUSD',
    'time': 1567428848005,
    'last_update': ...
  }
}
```

オープンポジション

このサブスクリプションフィードはユーザーアカウントのオープンポジションを発行します。

```
SubscribeOpenPositions();
```

後で、**UnSubscribeOpenPositions** メソッドを呼び出して `OpenPositions` からアンサブスクライブできます。

```
UnSubscribeOpenPositions();
```

サーバーからのレスポンス例

```
{
  "feed": "open_positions",
  "account": "DemoUser",
  "positions": [{
    "instrument": "fi_xbtusd_180316",
    "balance": 2000.0,
    "entry_price": 11675.86541981,
    "mark_price": 11090.0,
    "index_price": 12290.550000000001,
    "pnl": -0.00905299
  }]
}
```

アカウントログ

このサブスクリプションフィードはアカウント情報を配信します。

```
SubscribeAccountLog();
```

後で **UnSubscribeAccountLog** メソッドを呼び出すことで AccountLog のサブスクリプションを解除できます。

```
UnSubscribeAccountLog();
```

サーバーからのレスポンス例

```
{
  '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}]
}
```

約定

このサブスクリプションフィードはフィル情報を発行します。

```
SubscribeFills();
```

後で **UnSubscribeFills** メソッドを呼び出すことで Fills のサブスクライブを解除できます。

```
UnSubscribeFills();
```

サーバーからのレスポンス例

```
{  
  
  "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

このサブスクリプションフィードは、ユーザーのオープンオーダーに関する情報を配信します。

```
SubscribeOpenOrders();
```

後で、**UnSubscribeOpenOrders** メソッドを呼び出して OpenOrders のサブスクライブを解除できます

```
UnSubscribeOpenOrders();
```

サーバーからのレスポンス例

```
{  
  
  "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"
}
]
}
```

アカウント残高とマージン

このサブスクリプションフィードはクライアントのアカウントの残高とマージン情報を返します。

```
SubscribeAccountBalanceAndMargins();
```

後で **UnSubscribeAccountBalanceAndMargins** メソッドを呼び出してアカウント残高の購読を解除できます

```
UnSubscribeAccountBalanceAndMargins();
```

サーバーからのレスポンス例

```
{
"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
}
```

通知

このサブスクリプションフィードはクライアントに通知を発行します。

```
SubscribeNotifications();
```

後で **UnSubscribeNotifications** メソッドを呼び出して通知の購読を解除できます

```
UnSubscribeNotifications();
```

サーバーからのレスポンス例

```
{
"feed": "notifications_auth",
```



```
"notifications":[  
  {  
    "id":5,  
    "type":"market",  
    "priority":"low",  
    "note":"通知を説明するメモ。",  
    "effective_time":1520288300000  
  },  
  ...  
]  
}
```

API Kraken Futures | REST パブリック API

接続

URL: <https://futures.kraken.com/derivatives/api/v3>

Kraken Futures パブリック API は認証が不要です。

設定

唯一の設定は、REST HTTP リクエストのログを有効にするかどうかです。すべての HTTP リクエスト/レスポンスをテキストファイルに保存する場合は HTTPLogOptions を有効にします。

イベント

OnKrakenHTTPException: このイベントは、REST Api からHTTP要求を行う際に例外が発生した場合に呼び出されます。

メソッド

GetFeeSchedules

このエンドポイントはすべての手数料スケジュールを一覧表示します。認証は必要ありません。

```
KrakenFutures->REST_API->GetFeeSchedules();
```

Order Book

このエンドポイントは、現在上場されているすべての先物契約の非累積オーダーブック全体を返します。

```
KrakenFutures->REST_API->GetOrderBook("PI_XBTUSD");
```

ティッカー

このエンドポイントは、現在上場されているすべての Futures コントラクトとインデックスの現在の市場データを返します。

```
KrakenFutures->REST_API->GetTickers();
```

機器

このエンドポイントは、現在一覧表示されているすべての先物コントラクトとインデックスの仕様を返します。

```
KrakenFutures->REST_API->GetInstruments();
```

履歴

このエンドポイントは、指定されたlastTime値からの直近100件の取引を返します。値が指定されていない場合は直近100件の取引を返します。このエンドポイントは、呼び出された時点またはトレーディングエンジンの最終リリース以降(いずれか早い方)から 最大7日間の取引履歴のみを返します。

```
KrakenFutures->REST_API->GetHistory("PI_XBTUSD");
```

API Kraken Futures | REST プライベート API

接続

URL: <https://futures.kraken.com/derivatives/api/v3>

認証

REST Private API には API キーと API シークレットが必要です。これらの値はアカウントの Kraken によって提供されます。

```
Kraken->ApiKey = "api key";  
Kraken->ApiSecret = "api secret";
```

メソッド

EditOrderByOrderId

このエンドポイントを使用すると、現在リストされている Futures コントラクトの既存の注文を編集できます。

aOrderId: 編集したい注文の ID

aSize: 注文に関連するサイズ。

aLimitPrice: 注文に関連付けられた指値価格です。

aStopPrice: ストップ注文に関連するストップ価格。古い注文タイプが Stop の場合は必須です。

```
KrakenFutures->REST_API->EditOrderByOrderId("Order_Id", 2, 1000);
```

EditOrderByCliOrderId

このエンドポイントを使用すると、現在リストされている Futures コントラクトの既存の注文を編集できます。

aCliOrderId: ユーザーが指定した注文識別子。グローバルに一意である必要があります。

aSize: 注文に関連するサイズ。

aLimitPrice: 注文に関連付けられた指値価格です。

aStopPrice: ストップ注文に関連するストップ価格。注文タイプが Stop の場合は必須です。

```
KrakenFutures->REST_API->EditOrderByCliOrderId("Cli_Order_Id", 2, 1000);
```

SendMarketOrder

このエンドポイントを使用すると、成行注文を送信できます。

aSide: 注文の方向: 買いまたは売り。

aSymbol: 先物のシンボル

aSize: 注文に関連付けられたサイズ。

```
KrakenFutures->REST_API->SendMarketOrder(kosfBuy, "PI_XBTUSD", 1);
```

SendLimitOrder

このエンドポイントでは指値注文を送信できます。

aSide: 注文の方向: 買いまたは売り。

aSymbol: 先物のシンボル

aSize: 注文に関連付けられたサイズ。

aLimitPrice: 注文に関連付けられた指値価格です。

```
KrakenFutures->REST_API->SendLimitOrder(kosfBuy, "PI_XBTUSD", 1, 1000);
```

SendStopOrder

このエンドポイントを使用すると、Stop Orderを送信できます。

aSide: 注文の方向: 買いまたは売り。

aSymbol: 先物のシンボル

aSize: 注文に関連付けられたサイズ。

aStopPrice: ストップ注文に関連付けられたストップ価格。

aLimitPrice: 注文に関連付けられた指値価格です。

```
KrakenFutures->REST_API->SendStopOrder(kosfBuy, "PI_XBTUSD", 1, 1000, 900);
```

SendTakeProfitOrder

このエンドポイントを使用して利益確定注文を送信できます。

aSide: 注文の方向: 買いまたは売り。

aSymbol: 先物のシンボル

aSize: 注文に関連付けられたサイズ。

aStopPrice: ストップ注文に関連付けられたストップ価格。

aLimitPrice: 注文に関連付けられた指値価格です。

```
KrakenFutures->REST_API->SendTakeProfitOrder(kosfBuy, "PI_XBTUSD", 1, 1000, 900);
```

SendOrder

このエンドポイントを使用すると、現在上場されているFutures契約に対して、limit、stop、take profit、または immediate-or-cancel注文を送信できます。

OrderType: kotfLMT、kotfPOST、kotfMKT、kotfSTP、kotfTAKE_PROFIT、kotfIOC のいずれかを選択します。

Symbol: 先物のシンボル

Side: 注文の方向（買いまたは売り）。

Size: 注文に関連するサイズ。

StopPrice: ストップ注文に関連付けられたストップ価格。

LimitPrice: 注文に関連付けられた指値価格。

TriggerSignal: Stop または TakeProfit 注文を出す場合に使用するトリガーシグナル。kotsMark、kotsIndex、kotsLast のいずれかを選択してください。

CliOrderId: ユーザーが指定する注文識別子。グローバルに一意でなければなりません。

ReduceOnly: 既存のポジションを縮小のみ行う注文にする場合は true に設定します。既存のポジションを増加させる注文は拒否されます。デフォルトは false です。

```
TsgcHTTPKrakenFuturesOrder *oOrder = new TsgcHTTPKrakenFuturesOrder(this);
try
{
    oOrder->Side = kosfBuy;
    oOrder->Symbol = "PI_XBTUSD";
    oOrder->OrderType = kotfMKT;
    oOrder->Size = 1;
    KrakenFutures->REST_API->SendOrder(oOrder);
```

```
}  
_finally  
{  
    oOrder->Free;  
}
```

CancelOrderByOrderId

このエンドポイントは先物契約のオープンオーダーのキャンセルを可能にします。

aOrderId: 編集したい注文の ID

```
KrakenFutures->REST_API->CancelOrderByOrderId("Order_Id");
```

CancelOrderByCliOrderId

このエンドポイントは先物契約のオープンオーダーのキャンセルを可能にします。

aCliOrderId: ユーザーが指定した注文識別子。グローバルに一意である必要があります。

```
KrakenFutures->REST_API->CancelOrderByCliOrderId("Cli_Order_Id");
```

GetFills

このエンドポイントはすべての先物コントラクトの約定済み注文に関する情報を返します。

aLastFillDate: 指定されていない場合、任意のfutures契約の直近100件の約定を返します。指定された場合、lastFillTime以前の100件のエントリを返します。

```
KrakenFutures->REST_API->GetFills("2020-07-22T13:45:00.000Z");
```

転送

このエンドポイントを使用すると、同じ担保通貨を持つ2つのマージン口座間、またはマージン口座とキャッシュ口座間で資金を移動できます。

aFromAccount: 資金を移動するキャッシュまたはマージンアカウントの名前。

aToAccount: 資金を移動先のキャッシュまたはマージンアカウントの名前。

aUnit: 転送する単位。

aAmount: 転送する金額。

```
KrakenFutures->REST_API->Transfer("FI_XBTUSD", "cash", "xbt", 1.5);
```

GetOpenPositions

このエンドポイントは先物契約のすべてのオープンポジションのサイズと平均エントリー価格を返します。満期を迎えたが未決済の先物契約も含まれます。

```
KrakenFutures->REST_API->GetOpenPositions();
```

GetNotifications

このエンドポイントは、プラットフォームの通知を提供します。

```
KrakenFutures->REST_API->GetNotifications();
```

GetAccounts

このエンドポイントはすべての Kraken Futures アカウント（現金アカウントまたはマージンアカウント）に関する主要情報を返します。デジタル資産残高、銘柄残高、マージン要件、マージントリガー推定値、利用可能資金、オープンポジションの損益、ポートフォリオ価値などの補助情報が含まれます。

```
KrakenFutures->REST_API->GetAccounts();
```

CancelAllOrders

このエンドポイントは先物契約のオープンオーダーのキャンセルを可能にします。

Symbol: すべてのオープン注文をキャンセルする先物商品 (オプション)

```
KrakenFutures->REST_API->CancelAllOrders();
```

CancelAllOrdersAfter

このエンドポイントはネットワーク障害からクライアントを保護するためのデッドマンスイッチメカニズムを提供します。クライアントはタイムアウト (秒) 付きのリクエストを送信でき、タイムアウト時にすべてのクライアント注文をキャンセルするカウントダウンタイマーが起動します。

aTimeout: 秒単位で指定されたタイムアウト。

```
KrakenFutures->REST_API->CancelAllOrdersAfter(60);
```

GetOpenOrders

このエンドポイントはすべての先物契約のオープン注文に関する情報を返します。

```
KrakenFutures->REST_API->OpenOrders();
```

GetHistoricalOrders

このエンドポイントは、アカウントで行われた過去の注文を返します。

aSince: DateTime 以降

aBefore: 基準より前の日時です

aSort: 昇順の場合は「asc」、降順の場合は「desc」

aContinuationToken: 前のレスポンスから提供された継続トークン。次の利用可能な結果セットを返す呼び出しで使用できます。

```
KrakenFutures->REST_API->GetHistoricalOrders(Now, Now - 5);
```

GetHistoricalTriggers

このエンドポイントはアカウントで行われた過去のトリガーを返します。

aSince: DateTime 以降

aBefore: 基準より前の日時です

aSort: 昇順の場合は「asc」、降順の場合は「desc」

aContinuationToken: 前のレスポンスから提供された継続トークン。次の利用可能な結果セットを返す呼び出しで使用できます。

```
KrakenFutures->REST_API->GetHistoricalTriggers(Now, Now - 5);
```

GetHistoricalExecutions

このエンドポイントは、アカウントでの過去の約定の照会を可能にします。

aSince: DateTime 以降

aBefore: 基準より前の日時です

aSort: 昇順の場合は「asc」、降順の場合は「desc」

aContinuationToken: 前のレスポンスから提供された継続トークン。次の利用可能な結果セットを返す呼び出しで使用できます。

```
KrakenFutures->REST_API->GetHistoricalExecutions(Now, Now - 5);
```

WithdrawalToSpotWallet

このエンドポイントを使用すると、Kraken FuturesウォレットからあなたのKraken Spotウォレットへのデジタル資産の出金リクエストを送信できます。

aCurrency: 引き出すデジタル資産（例: xbt または xrp）。

aAmount: 出金する通貨の金額。

```
KrakenFutures->REST_API->WithdrawalToSpotWallet("xbt", 1000);
```

GetFeeScheduleVolumes

このエンドポイントは 30 日間の USD 取引量を返します。

```
KrakenFutures->REST_API->GetFeeScheduleVolumes();
```

GetAccountLogCSV

このエンドポイントにより、クライアントはアカウントログの CSV ファイルをダウンロードできます。

```
KrakenFutures->REST_API->GetAccountLogCSV();
```


API Pusher

Pusher

Pusherは、WebSocketプロトコルに基づいた優れた機能を持つ使いやすく信頼性の高いプラットフォームです。柔軟なpub/subメッセージング、ライブユーザーリスト（プレゼンス）、認証などの機能を提供します。

Pusher WebSocket API は7です。

データは UTF8 エンコードされた JSON を含むテキストデータとして WebSocket 経由で双方向に送信されます（バイナリ WebSocket フレームはサポートされていません）。

サーバーへの接続をテストするには **Ping** メソッドを呼び出すことができます。基本的に、相手側から受信したメッセージはすべて接続が生きていることを意味します。メッセージがない場合、どちらの側もピングメッセージを送信することで相手が応答しているか確認でき、相手側はポンで応答するはずです。

接続する前に、以下のフィールドを入力する必要があります：

```
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)
```

重要

Pusher では、WebSocket クライアントが前述のフィールド（キー、クラスターなど）を使用して構築された URL に接続する必要があります。これらのフィールドは、Pusher コンポーネントにクライアントを割り当てるときに URL の構築に使用されます。URL が正しく構築されるよう、Pusher 設定フィールドを入力した後にクライアントを設定してください。以下に擬似コードを示します：

```
// pusher フィールドを設定
pusher.cluster = ...
pusher.key = ...
// set client
pusher.client = websocket クライアント
// 接続を開始
websocket client.Active = true;
```

接続が成功すると、**OnPusherConnect** イベントが発生し、以下のフィールドが取得されます：

- ソケット ID：接続されたクライアントの一意の識別子。
- Timeout: クライアントが ping メッセージを開始するまでのサーバー非アクティブ秒数 (コンポーネントが自動的に処理します)。

エラーが発生すると **OnPusherError** が発生し、エラーに関する情報が提供されます。エラーは、無効な認証、無効なコマンドなどに応答して Pusher から送信される場合があります。

4000-4099

Pusherによって接続が閉じられるエラーであり、同じパラメータを使用して再接続を試みても成功しないことを示します。

- 4000: アプリケーションはSSL接続のみを受け付けます。wss:// を使用して再接続してください。
- 4001: アプリケーションが存在しません
- 4003: アプリケーションが無効になっています
- 4004: アプリケーションが接続クォータを超えています。
- 4005: パスが見つかりません
- 4006: バージョン文字列の形式が無効です
- 4007: サポートされていないプロトコルバージョン
- 4008: プロトコルバージョンが指定されていません

4100-4199

Pusher によって接続が閉じられたエラーを示しており、クライアントは 1 秒以上後に再接続できます。

- 4100: 容量オーバー

4200-4299

Pusher によって接続が閉じられるエラーが発生したことを示します。クライアントはすぐに再接続できます。

- 4200: すぐに汎用再接続
- 4201: Pong 返信が受信されませんでした: ping がクライアントに送信されましたが、返信が受信されませんでした。ping および pong メッセージを参照してください。
- 4202: 非アクティブにより切断：クライアントが長時間非アクティブで（現在24時間）、クライアントが ping をサポートしていません。新しい WebSocket ドラフトにアップグレードするか、このプロトコルのバージョン5以上を実装してください。

4300-4399

その他の種類のエラー。

4301: レートリミットによりクライアントイベントが拒否されました

チャンネル

チャンネルは Pusher の基本的な概念です。各アプリケーションには複数のチャンネルがあり、各クライアントはサブスクライブするチャンネルを選択できます。

チャンネルは以下を提供します:

- データをフィルタリングする方法。例えばチャットアプリケーションでは、「犬」について議論したい人々のためのチャンネルがあるかもしれません。
- 異なる情報ストリームへのアクセスを制御する方法。例えば、プロジェクト管理アプリケーションでは「projectX」の更新を取得するためにユーザーを認可したい場合があります。

チャンネルを使用してデータをフィルタリングすることを強くお勧めします。イベントを使用してフィルタリングするべきではありません。これは、チャンネルにパブリッシュされたすべてのイベントが、イベントバインディングに関わらずすべてのサブスクライバーに送信されるためです。

チャンネルを明示的に作成する必要はなく、クライアントの要求に応じてインスタンス化されます。これは、チャンネルの作成が簡単であることを意味します。クライアントにそれをサブスクライブするよう伝えるだけです。

以下の種類のチャンネルがサポートされています。

- パブリックチャンネル**は、名前を知っている人であれば誰でもサブスクライブできます
- プライベートチャンネル**は、ブロードキャストするデータへのアクセスをサーバーが制御できるメカニズムを導入します。
- Presence チャンネル**はプライベートチャンネルの拡張です。サブスクリプション時にユーザー情報を登録でき、チャンネルの他のメンバーに誰がオンラインかを知らせることができます。
- キャッシュチャンネル**は最後にトリガーされたイベントを記憶し、新しいサブスクライバーへの最初のイベントとして送信します (パブリック、プライベート、プレゼンスの各バリエーション)
- Private-Encrypted チャンネル**は、NaCl secretbox を使用してエンドツーエンド暗号化を提供し、Pusher でさえメッセージデータを読み取れないようにします

パブリックチャンネル

パブリックチャンネルはサブスクライブするための認証を必要としないため、公開アクセス可能なデータに使用してください。

いつでもチャンネルにサブスクライブおよびサブスクライブ解除できます。Pusher の接続が完了するまで待つ必要はありません。

例: チャンネル「my-channel」にサブスクライブします。

```
CBuilder
APIPusher->Subscribe("my-channel");
```

サブスクライブが成功した場合、**OnPusherSubscribe** イベントが発生します。エラーがある場合は **OnPusherError** イベントでメッセージを受け取ります。

サブスクライブされたチャンネルからのすべてのメッセージは **OnPusherEvent** イベントで受信されます。

Publish メソッドが呼び出され、チャンネルがパブリックの場合、コンポーネントは WebSocket プロトコルを使用する代わりに HTTP プロトコルを使用し、TriggerEvent メソッドを呼び出します (WebSocket プロトコルを使用したパブリッシュは許可されていません)。

プライベートチャンネル

Indy 10.5.7 以降が必要です

プライベートチャンネルは、チャンネルへのアクセスを何らかの方法で制限する必要がある場合に使用します。ユーザーがプライベートチャンネルをサブスクライブするには、許可の認可が必要です。

使用例: チャンネル「my-private-channel」を購読します。

```
CBuilder
APIPusher->Subscribe("my-private-channel", pscPrivateChannel);
```

サブスクライブが成功した場合、**OnPusherSubscribe** イベントが発生します。エラーがある場合は **OnPusherError** イベントでメッセージを受け取ります。

サブスクライブされたチャンネルからのすべてのメッセージは **OnPusherEvent** イベントで受信されます。

Presence チャンネル

Indy 10.5.7 以降が必要です

プレゼンスチャンネルはプライベートチャンネルのセキュリティを基盤として構築され、そのチャンネルを購読しているユーザーを把握するための追加機能を公開します。これにより、チャットルームや「オンラインユーザー」タイプの機能をアプリケーションに簡単に構築できます。チャットルーム、ドキュメントのコラボレーター、同じ Web ページを閲覧しているユーザー、ゲームの対戦相手などを考えてみてください。

Presence チャンネルはプライベートチャンネルと同じ方法でクライアント API からサブスクライブしますが、チャンネル名には `presence-` のプレフィックスを付ける必要があります。プライベートチャンネルと同様に、現在のユーザーがチャンネルへのアクセス権を持つかどうかを確認するために、設定可能な認証 URL に HTTP リクエストが送信されます。

プレゼンスチャンネルにバインドされたイベントにバインドすることで、チャンネルにサブスクライブおよびアンサブスクライブするユーザーの情報にアクセスでき、`channel.members` プロパティでチャンネルにサブスクライブしているユーザーの現在の状態が利用できます。

使用例: チャンネル `"my-presence-channel"` にサブスクライブします。

```
APIPusher->Subscribe("my-presence-channel", pscPresenceChannel,
    '{"user_id":"","user_info":{"name":"","John Smith"}}');
```

サブスクライブが成功した場合、**OnPusherSubscribe** イベントが発生します。エラーがある場合は **OnPusherError** イベントでメッセージを受け取ります。

サブスクライブされたチャンネルからのすべてのメッセージは **OnPusherEvent** イベントで受信されます。

キャッシュチャンネル

キャッシュチャンネルは最後にトリガーされたイベントを記憶し、これを最初のイベントとして新しいサブスクライバーに送信します。

イベントがキャッシュチャンネルでトリガーされると、Pusher Channelsはこのイベントをキャッシュし、クライアントがキャッシュチャンネルをサブスクライブしたときに、キャッシュされた値が存在すれば、それがそのチャンネルの最初のイベントとしてクライアントに送信されます。この動作により、開発者は他の場所から取得するための追加のロジックを追加せずに初期状態を提供できます。

以下のキャッシュチャンネルがサポートされています:

- パブリックキャッシュチャンネル
- プライベートキャッシュチャンネル
- Presence キャッシュチャンネル

例: パブリックキャッシュチャンネル `"my-cache-channel"` にサブスクライブします。

```
APIPusher->Subscribe("my-cache-channel", pscCacheChannel);
```

サブスクライブが成功した場合、**OnPusherSubscribe** イベントが発生します。エラーがある場合は **OnPusherError** イベントでメッセージを受け取ります。

サブスクライブされたチャンネルからのすべてのメッセージは **OnPusherEvent** イベントで受信されます。

cacheチャンネルにサブスクライブする際にキャッシュされたイベントが存在しない場合、**OnPusherCacheMiss** イベントが発生し、チャンネル名が提供されます。これにより、キャッシュされたデータが利用できない場合をアプリケーションで処理できます。

プライベート暗号化チャンネル

プライベート暗号化チャンネルは、メッセージのエンドツーエンド暗号化を提供します。プライベートチャンネルと同様に認証が必要ですが、さらに NaCl secretbox を使用してすべてのデータペイロードが暗号化されるため、認可されたサブスクライバーのみがコンテンツを読み取ることができます。Pusher 自身もメッセージを復号化できません。

private-encryptedチャンネルを使用するには、認証中に **SharedSecret** を提供する必要があります。共有シークレットは、メッセージデータの暗号化と復号化に使用されます。

使用例: プライベート暗号化チャンネル `"my-encrypted-channel"` をサブスクライブします。

```
APIPusher->Subscribe("my-encrypted-channel", pscPrivateEncryptedChannel);
```

暗号化とキャッシュチャンネルの動作を組み合わせたプライベート暗号化キャッシュのバリエーションも利用可能です。

```
APIPusher->Subscribe("my-encrypted-cache-channel", pscPrivateEncryptedCacheChannel);
```

プライベート暗号化チャンネルで **OnPusherAuthentication** イベントを使用する場合、応答オブジェクトの **SharedSecret** プロパティに暗号化キーを設定できます。

```
void OnPusherAuthenticationEvent(TObject *Sender,  
    TsgcWSPusherRequestAuthentication *AuthRequest,  
    TsgcWSPusherResponseAuthentication *AuthResponse)  
{  
    AuthResponse->SharedSecret = "your-shared-secret-key";  
}
```

プレゼンスイベント

Presenceチャンネルは、ユーザーがチャンネルに参加または退出したときにアプリケーションに通知する追加イベントを提供し、サブスクリプション数を追跡できるようにします。

OnPusherMemberAdded

新しいメンバーがプレゼンスチャンネルをサブスクライブしたときに発生します。チャンネル名、ユーザー ID、参加したメンバーのユーザー情報を提供します。

```
void PUSHERPusherMemberAdded(TObject *Sender,  
    String Channel, String UserId, String UserInfo)  
{  
    Log("Member joined: " + UserId + " on " + Channel);  
}
```

OnPusherMemberRemoved

メンバーがプレゼンスチャンネルから購読解除したときに発生します。退出したメンバーのチャンネル名、ユーザー ID、ユーザー情報を提供します。

```
void PUSHERPusherMemberRemoved(TObject *Sender,  
    String Channel, String UserId, String UserInfo)  
{  
    Log("Member left: " + UserId + " on " + Channel);  
}
```

OnPusherSubscriptionCount

チャンネルのサブスクリプション数が変更されたときに発生します。チャンネル名と現在のサブスクライバー数を提供します。このイベントは Pusher ダッシュボードで有効にする必要があります。

```
void PUSHERPusherSubscriptionCount(TObject *Sender,  
    String Channel, int SubscriptionCount)  
{  
    Log(Channel + " has " + IntToStr(SubscriptionCount) + " subscribers");  
}
```

OnPusherCacheMiss

キャッシュされたイベントがないキャッシュチャンネルにサブスクライブするときに発生します。チャンネル名を提供します。これにより、キャッシュされたデータが利用できない場合に、アプリケーションが別のソースからデータをフェッチするなどして対処できます。

```
void PUSHERPusherCacheMiss(TObject *Sender, String Channel)  
{  
    Log("Cache miss on: " + Channel);  
}
```

メッセージのパブリッシュ

サブスクライブしたチャンネルからメッセージを受信できるだけでなく、他のサブスクライブ中のユーザーにメッセージを送信することもできます。

チャンネルのすべてのサブスクライブユーザーにメッセージを送信するには **Publish** メソッドを呼び出してください。

例: "my-channel" のすべてのサブスクライブユーザーにイベントを送信する

```
APIPusher->Publish("my-event", "my-channel");
```

クライアント（接続）ごとに1秒あたり最大10件のメッセージを公開してください。このレート制限を超えてトリガーされたイベントは Pusher API によって拒否されます。これはシステムの問題ではなく、クライアントの問題です。このレートでメッセージを送信するチャンネル内の100人のクライアントは、それぞれ1秒あたり1,000件のメッセージを処理する必要があります！一部の最新ブラウザーはこれを処理できるかもしれませんが、おそらく良いアイデアではありません。

REST API

API は <http://api-CLUSTER.pusher.com> でホストされています。CLUSTER はご自身のアプリのクラスター（例：eu）に置き換えてください。

HTTP ステータスコードはリクエストの成功または失敗を示すために使用されます。以下のステータスが一般的です：

200 リクエスト成功。本文にはレスポンスデータの JSON ハッシュが含まれます

400 エラー: レスポンスボディに詳細あり

401 認証エラー: レスポンスボディに説明が含まれます

403 Forbidden: アプリが無効またはメッセージクォータを超過しています。

以下の REST API 関数が実装されています。

Function	説明
TriggerEvent	指定されたチャンネルで新しいイベントをトリガーします。オプションの SocketId (クライアントを除外するため) と Info パラメータをサポートします。
TriggerBatchEvents	単一の HTTP リクエストで複数のイベントを発火します。イベントオブジェクトの JSON 配列を受け入れます。
GetChannels	すべてのアクティブなチャンネルのリストを提供します。オプションの FilterByPrefix と Info パラメータをサポートします。
GetChannel	特定のチャンネルに関する情報を提供します。オプションの Info パラメータをサポートします。
GetUsers	チャンネルに接続しているすべてのユーザーのリストを提供します。
TerminateUserConnections	ユーザー ID によって特定のユーザーのすべての接続を終了します。

TriggerEvent

1つ以上のチャンネルでイベントをトリガーします。イベント名、チャンネル名、データペイロードが必要です。

パラメータ	説明
aEventName	トリガーするイベントの名前。
aChannel	イベントをトリガーするチャンネル名。
aData	イベントデータ (JSON 文字列)。
aSocketId (オプション)	イベントの受信から除外するソケット ID。送信者が自分のメッセージを受信しないようにするのに便利です。
alInfo (オプション)	レスポンスに含める属性のカンマ区切りリスト (例: "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

単一の API 呼び出しで複数のイベントをトリガーします。各イベントに対して個別のリクエストを行うよりも効率的です。batch パラメーターは、各オブジェクトに「channel」、「name」、「data」フィールドを持つイベントオブジェクトの配列を含む JSON 文字列でなければなりません。

```
APIPusher->TriggerBatchEvents(
    "[{"channel\":\"my-channel\",\"name\":\"my-event\",\"data\":\"hello\"}, {"channel\":\"my-channel-2\",\"name\":\"my-event\",\"data\":\"world\"}]");
```

GetChannels

アクティブなチャンネルの一覧を返します。結果をフィルタリングし、追加情報をリクエストするためのオプションパラメータをサポートします。

パラメータ	説明
aFilterByPrefix (オプション)	名前プレフィックスでチャンネルをフィルタリングします (例: プレゼンスチャンネルのみを一覧表示するには "presence-")。
alInfo (オプション)	レスポンスに含める属性のカンマ区切りリスト (例: 「user_count」)。

```
// get all channels
APIPusher->GetChannels();

// get only presence channels with user count
APIPusher->GetChannels("presence-", "user_count");
```

GetChannel

特定のチャンネルに関する情報を返します。

パラメータ	説明
aChannel	情報を取得するチャンネル名。
alInfo (オプション)	含める属性のカンマ区切りリスト (例: "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 チャンネルに接続しているユーザーのリストを返します。チャンネル名には完全なプレフィックス（例: "presence-my-channel"）を含める必要があります。

```
APIPusher->GetUsers("presence-my-channel");
```

TerminateUserConnections

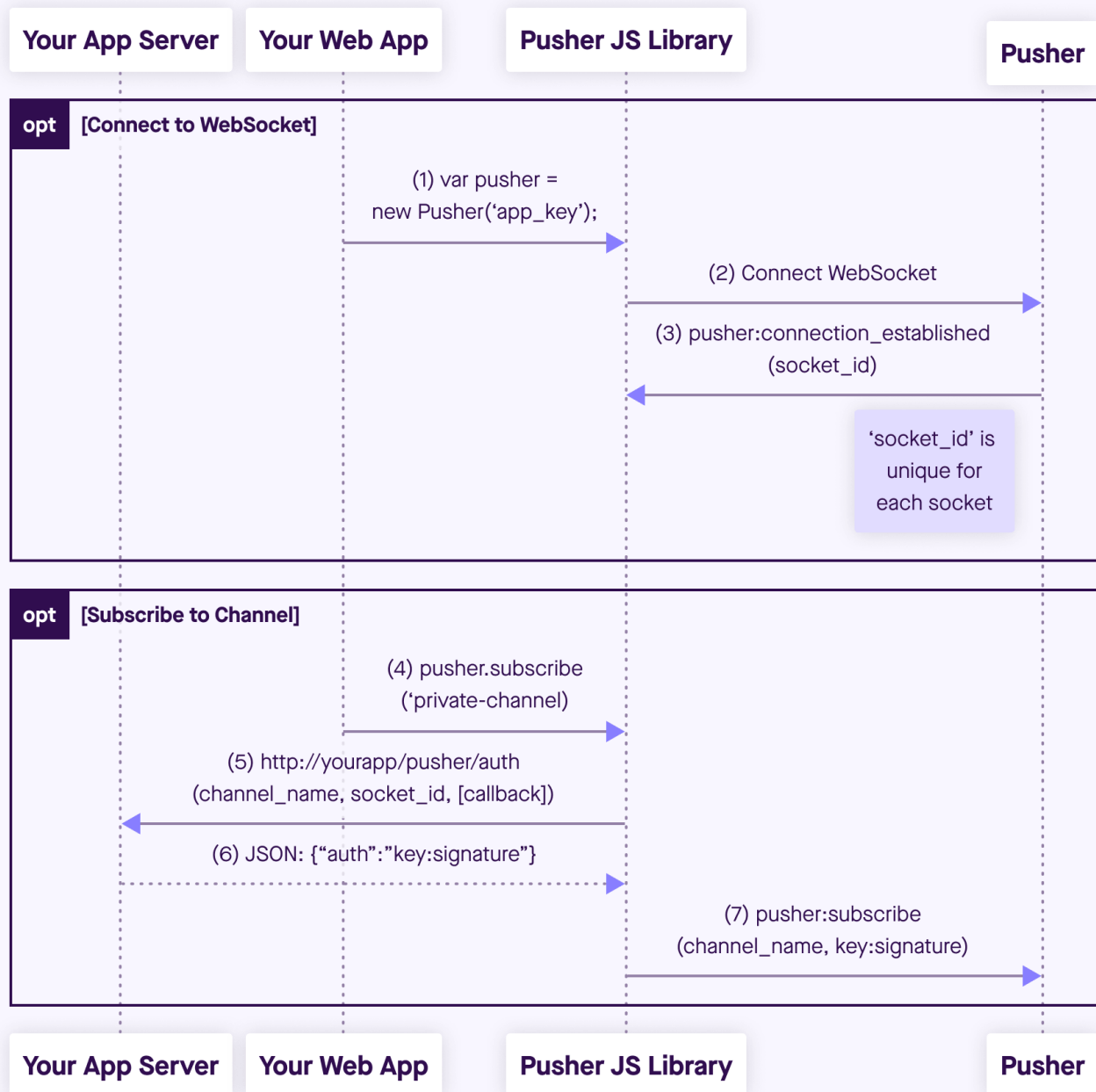
特定のユーザーによって確立されたすべての接続を終了します。特定のユーザーをすべてのチャンネルから強制的に切断するために使用できます。ユーザー ID は、ユーザーがプレゼンスチャンネルをサブスクライブしたときに使用した「user_id」と一致する必要があります。

```
APIPusher->TerminateUserConnections("1234");
```

カスタム認証

Pusher はプライベートチャンネルまたはプレゼンスチャンネルへのサブスクリプションのみを許可しており、接続が認証トークンを提供している場合はアクセスを制限できます。

OnPusherAuthentication イベントを使用して独自の認証フローを構築できます。このイベントはサブスクリプションメッセージが Pusher が提供する秘密鍵で署名される前に呼び出されます。このイベントには、SocketId、チャンネル名などのフィールドを持つリクエスト認証の 2 つのパラメーターがあります。これらは独自の認証サーバーがリクエストを認証するかどうかで使用できます。Pusher の認証フローを示すスクリーンショットを以下に示します



クライアントがPusherサーバーに接続すると、Pusherが提供するキーを送信し、サーバーは識別IDを返します (`socket_id`)。

クライアントがプライベート (またはプレゼンス) チャンネルにサブスクライブする場合、sgcWebSockets クライアントは Pusher が提供するシークレットキーを使用してサブスクリプションメッセージに含まれる署名を作成します。OnPusherAuthentication イベントを使用すると、メッセージに署名するために必要なフィールドをキャプチャし、独自の認証メソッドを実装し、成功した場合に署名を返すことができます。この署名はサブスクリプションメッセージに含まれてサーバーに送信されます。

例:

```

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();
}
}
```

署名のフォーマットは次のとおりです:

プライベートチャンネル: key:HMAC256(SocketID, ChannelName)

プレゼンスチャンネル: キー: HMAC256(SocketID, ChannelName, Data)

TsgcWSPusherResponseAuthentication オブジェクトは次のプロパティを提供します:

プロパティ	説明
シークレット	署名の計算に使用される Pusher のシークレットキー。Pusher.Secret が設定されている場合はあらかじめ入力されます。
署名	計算された認証署名。空のままにすると、コンポーネントがSecretを使用して自動的に計算します。
SharedSecret	プライベート暗号化チャンネルの共有シークレットキー。pscPrivateEncryptedChannel または pscPrivateEncryptedCacheChannel にサブスクライブする場合に必要です。メッセージデータのエンドツーエンド暗号化に使用されます。

API Bitmex

Bitmex

Bitmex は暗号通貨取引所およびデリバティブ取引プラットフォームです。

次のAPIがサポートされています。

1. **WebSocketストリーム**: 一部のメソッドを購読してリアルタイムでデータを取得できます。イベントはサーバーから購読者であるクライアントにプッシュされます。プロトコルとしてWebSocketを使用します。
2. **REST API**: クライアントはサーバーの市場データやアカウントデータをリクエストできます。認証にはAPIキーとシークレットが必要で、プロトコルとしてHTTPSを使用します。

プロパティ

Bitmex API には public と private の2種類のメソッドがあります。パブリックメソッドはティック価格の取得など、認証なしでアクセスできます。一部はプライベートであり、ユーザーデータに関連するため、Bitmex APIキーの使用が必要です。

- **ApiKey**: Bitmex アカウントで新しいAPI キーをリクエストできます。値をこのプロパティにコピーしてください。
- **ApiSecret**: API のシークレット。安全に保管してください。
- **TestNet**: 有効にすると、Bitmex Demo Account に接続します（デフォルトはfalse）。
- **HTTPLogOptions**: HTTP リクエストのログをテキストファイルに保存します。
 - **Enabled**: 有効の場合、WebSocket API のすべての HTTP リクエストを保存します。
 - **FileName**: ログが保存されるファイルのフルパス

最も一般的な使用方法

- **WebSockets API**
 - [WebSocket APIへの接続方法](#)
 - [WebSocketチャンネルのサブスクライブ方法](#)
- **REST API**
 - [Bitmex 注文の発注方法](#)

WebSocket API

サブスクライブ / アンサブスクライブ

BitMEX はリアルタイムデータへのサブスクライブをサポートしています。このアクセスは接続後にレートが制限されず、プログラムに最新のデータを取得するための最善の方法です。一部のトピックでは、取引、クオートなどのイベントをシンボルでフィルタリングするためにシンボルを渡すことができます。

以下のサブスクリプショントピックは認証なしで利用可能です:

- **btmAnnouncement**: サイトアナウンス
- **btmChat**: Trollbox チャット
- **btmConnected**: 接続されているユーザー/ボットの統計
- **btmFunding**: スワップファンディングレートの更新。ファンディング間隔ごとに送信されます（通常 8 時間）。
- **btmInstrument**: 売買高と買値/売値を含むインストゥルメントの更新
- **btmInsurance**: 毎日の保険基金の更新
- **btmLiquidation**: 板に入力された清算注文
- **btmOrderBookL2_25**: レベル 2 オーダーブックのトップ 25 レベル

- **btmOrderBookL2**: フルレベル 2 オーダーブック
- **btmOrderBook10**: 従来のフルブックプッシュを使用した上位10レベル
- **btmPublicNotifications**: システム全体の通知（短期間のメッセージに使用）
- **btmQuote**: 板の最上位
- **btmQuoteBin1m**: 1分間の相場ビン
- **btmQuoteBin5m**: 5 分間のクオートビン
- **btmQuoteBin1h**: 1 時間クオートビン
- **btmQuoteBin1d**: 1日のクオートビン
- **btmSettlement**: 決済
- **btmTrade**: ライブ取引
- **btmTradeBin1m**: 1分間の取引ビン
- **btmTradeBin5m**: 5分間の取引ビン
- **btmTradeBin1h**: 1 時間の取引ビン
- **btmTradeBin1d**: 1日の取引ビン

以下のサブジェクトには認証が必要です：

- **btmAffiliate**: 総紹介ユーザー数や支払いパーセンテージなどのアフィリエイトステータス
- **btmExecution**: 個別の約定。1 注文に対して複数発生する場合があります
- **btmOrder**: 注文に関するライブ更新
- **btmMargin**: 現在の口座残高と証拠金要件に関する更新
- **btmPosition**: ポジションの更新
- **btmPrivateNotifications**: 個別通知（現在は未使用）
- **btmTransact**: 入出金の更新
- **btmWallet**: 総預金 & 出金を含む Bitcoin アドレス残高データ

受信したメッセージの例：

```
{
  "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},
  ]
}
```

認証

ユーザーロックされたストリームにサブスクライブする場合は、最初に認証する必要があります。無効な認証は接続を閉じることに注意してください。

BitMEX API の使用には API キーが必要です。

永続的な API キーは IP アドレス範囲にロックでき、メインの認証情報を侵害することなくいつでも失効できます。また更新も不要です。

API キー認証を使用するには、アカウントで API キーを生成する必要があります。

認証が必要なトピックをサブスクライブする前に、**Authenticate**メソッドを呼び出してください。

CancelAllAfter (デッドマンズスイッチ)

CancelAllAfterメソッドは、デッドマンズスイッチ機能を実装します。タイムアウト値（ミリ秒単位）を指定して呼び出すと、タイムアウト期間内に後続のCancelAllAfter呼び出しを受信しない場合、すべての未約定注文をキャンセルするようサーバーに指示します。これは、ネットワーク切断時に注文がキャンセルされることを保証するのに便利です。

REST API

メソッド	説明
GetExecutions	これは、注文の開始とキャンセル、および注文ステータスの変更を含む、すべての生のトランザクションを返します。
GetExecutionsTradeHistory	これにより、より焦点を絞ったトランザクションが返されます。
GetInstruments	これにより、決済済みまたは非上場のものを含む、すべての商品とインデックスが返されます。個々の商品を照会したり、複雑なフィルターを使用したりする場合は、このエンドポイントを使用してください。
GetOrders	オープン注文のみを取得するには
PlaceOrder	TsgcHTTPBitmexOrder オブジェクトを使用してローの注文を発注します。
PlaceMarketOrder	新しいMARKET注文を発注します。
PlaceLimitOrder	新しい LIMIT 注文を発注します。
PlaceStopOrder	新しいSTOP注文を発注します。
PlaceStopLimitOrder	新しい STOPLIMIT 注文を発注します。
AmendOrder	既存の注文を変更します。
CancelOrder	アクティブな注文をキャンセルします。
CancelAllOrders	すべてのアクティブな注文をキャンセルします。
CancelAllOrdersAfter	一定時間後にすべての注文をキャンセルします。
ClosePosition	オープンポジションをクローズします。
GetOrderBook	縦形式で現在のオーダーブックを取得する
GetPosition	ポジションを取得します。
SetPositionIsolate	ポジションごとに分離マージンまたはクロスマージンを有効にします。
SetPositionLeverage	ポジションごとにレバレッジを選択します。
SetPositionRiskLimit	リスク上限を更新します。
SetPositionTransferMargin	ポジションへのエクイティを振り込むまたは引き出す。
GetQuotes	クォートを取得します。
GetTrades	Get Trades
GetFunding	ファンディングデータを取得します。
GetInsurance	保険基金データを取得します。

GetTradeBucketed	設定可能なビンサイズでバケット化されたトレードデータ（OHLCV）を取得します。
GetQuoteBucketed	設定可能なビンサイズでバケット化されたクォートデータを取得します。
GetSettlement	決済データを取得します。
GetLiquidation	清算注文を取得します。
GetInstrumentIndices	機器のインデックスを取得します。
GetInstrumentCompositeIndex	インストゥルメントの複合インデックスデータを取得します。
GetStats	取引所全体の統計を取得します。
GetStatsHistory	過去の取引所統計を取得します。
GetStatsHistoryUSD	USD の過去の為替統計を取得します。
GetUserMargin	アカウントのマージンデータを取得します。
GetUserWallet	ウォレット情報を取得します。
GetUserWalletHistory	ウォレットのトランザクション履歴を取得します。
GetUserWalletSummary	ウォレットのサマリーを取得します。

Bitmex | Connect WebSocket API

Bitmex WebSocket API に接続するには、新しい Bitmex API クライアントを作成して TsgcWebSocketClient にアタッチするだけです。
以下の例をご覧ください:

```
TsgcWebSocketClient oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bitmex oBitmex = new TsgcWSAPI_Bitmex();
oBitmex->Client = oClient;
oClient->Active = true;
```

Bitmex | WebSocket チャンネルへの購読

Bitmexは、市場データや注文などのリアルタイム更新を取得するためにサブスクライブできるさまざまなチャンネルを提供しています... 以下に、Trade Channelへのサブスクライブ方法のサンプルを示します:

```
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 | 注文方法

Bitmex REST API にはパブリックエンドポイントとプライベートエンドポイントがあります。プライベートエンドポイントはトランザクションのセキュリティを高めるためにメッセージの署名が必要です。まず Bitmex アカウントにログインして新しい API を作成すると、以下の値が取得されます。

- ApiKey
- ApiSecret

これらのフィールドは Bitmex API クライアントコンポーネントの Bitmex プロパティで設定する必要があります。設定後、Bitmex REST API にプライベートリクエストを開始できます。

注文タイプ

すべての注文にはシンボルが必要です。それ以外のフィールドは、特に指定がない限りオプションです。有効な ordTypes は以下の通りです。

- **Limit**: デフォルトの注文タイプ。orderQty と price を指定します。
- **Market**: 従来の成行注文。成行注文は約定するか、破産価格に達するまで実行され、その時点でキャンセルされます。
- **Stop**: ストップマーケット注文。orderQty と stopPx を指定します。stopPx に達すると注文が板に入ります。
 - 売り注文の場合、トリガー価格が stopPx より低い場合に注文がトリガーされます。買いの場合は高い場合です。
 - 注意: ストップ注文はトリガーされるまでマージンを消費しません。完全にトリガーできるよう、アカウントに必要なマージンが利用可能であることを確認してください。
 - クローズストップには orderQty は必要ありません。下記の実行指示を参照してください。
- **StopLimit**: ストップマーケットに似ていますが、マーケット注文の代わりに指値注文を出します。orderQty、stopPx、price を指定してください。
- **MarketIfTouched**: Stop に似ていますが、トリガーは逆方向に行われます。利益確定注文に便利です。
- **LimitIfTouched**: 上記と同様。利益確定指値注文に使用します。
- **Pegged**: ペッグ注文は、ユーザーが現在の市場価格に対する相対的な指値価格を送信できます。pegPriceType と pegOffsetValue を指定してください。
 - ペッグ注文の execInst は Fixed である必要があります。これは、指値価格が注文受諾時に設定され、参照価格が変わっても変更されないことを意味します。
 - PrimaryPeg: 価格はニアタッチ価格に対して設定されます。
 - MarketPeg: 価格はファータッチ価格に対して設定されます。
 - ordType なしで送信された pegPriceType はペッグ注文として扱われます。

実行指示

以下の execInst がサポートされています。複数使用する場合はカンマで区切ります (例: LastPrice,Close)。

- **ParticipateDoNotInitiate**: Post-Only注文としても知られています。この注文が発注時に約定する場合、代わりにキャンセルされます。これは、注文が転送中に反対側のタッチがあなたに向かって移動するのを防ぐことを目的としています。送信後に反対側のタッチが離れることに賭けることを目的としたものではありません。当社はそのような行為を不正と見なし、監視しています。
- **MarkPrice, LastPrice, IndexPrice**: 逆指値注文およびイフタッチ注文のトリガー価格の決定に使用されます。いずれか一つのみ使用してください。デフォルトでは MarkPrice が使用されます。ペッグ注文の LastPeg 値を定義するためにも使用されます。
- **ReduceOnly**: ReduceOnly 注文は既存のポジションを縮小のみ可能で、増加はできません。注文が板で待機中に他の注文によってポジションが縮小された場合、ReduceOnly 指値注文の数量は調整または取り消されます。複数の ReduceOnly 注文がある場合、最も攻撃的でないものが最初に調整されます。
- **Close**: CloseはReduceOnlyを意味します。Close注文は、オープン数量が現在のポジションを超える場合、同じ側とシンボルを持つ他のアクティブなlimit注文をキャンセルします。これはストップに便利です。これらの注文をキャンセルすることで、Close Stopは実行に必要な証拠金を確保でき、ポジションの全サイズまでしか実行できません。orderQtyが指定されていない場合、Close注文のorderQtyは現在のポジションのサイズに等しくなります。

- orderQtyのないClose注文には側が必要であることに注意してください。これにより、BitMEXはstopPxの上下どちらでトリガーすべきかを判断できます。
- **LastWithinMark:** 以下の条件を満たす場合にのみストップトリガーを許可するために、LastPriceを使用するストップ注文で使用されます:
 - 売りストップマーケット / ストップリミット注文の場合
 - 最終価格 ≤ ストップ価格
 - 最終価格 ≥ マーク価格 × (1 - 5%)
 - 買いストップマーケット / ストップリミット注文の場合:
 - 最終価格 ≥ ストップ価格
 - 最終価格 ≤ マーク価格 × (1 + 5%)
- **固定 (Fixed) :** ペグ注文はexecInstをFixedにする必要があります。これはリミット価格が注文受け付け時に設定され、参照価格が変化しても変わらないことを意味します。

ペグ注文

ペグ注文では、ユーザーは現在の市場価格に対して相対的な指値価格を提出できます。指値価格は注文提出時に1回設定され、参照価格とともに変化しません。この注文タイプは提出後に遠い側が動くことを投機することを意図していません。私たちはそのような動作を悪用と見なし監視しています。

Peggedオーダーは、ordTypeがPegged、execInstがFixedです。

pegPriceTypeとpegOffsetValueも送信する必要があります。

- PrimaryPeg - 価格はニアタッチ価格に対して相対的に設定されます
- MarketPeg - 価格はファータッチ価格に対して相対的に設定されます

トレーリングストップペグ注文

トレーリングストップを作成するには TrailingStopPeg の pegPriceType を使用します。

価格は提出時に設定され、基礎となる価格（最終/マーク/インデックス）が0.1%以上移動した場合は1秒ごとに更新されます。stopPx は市場がペグから離れるにつれて移動し、市場がペグに近づくとき固定されます。

pegOffsetValue を使用して注文の stopPx を設定します。ペグは execInst で指定されたトリガー価格（デフォルトはMarkPrice）に設定されます。ストップセルおよびバイイフタッチオーダーには負のオフセットを使用します。

ordTypeが必要です: Stop、StopLimit、MarketIfTouched、LimitIfTouched。

トレーリングストップ

トレーリングストップを作成するには、'TrailingStopPeg'のpegPriceTypeを使用できます。ペグされたstopPxは、市場がペグから離れる方向に動くと一緒に動き、市場がペグに近づく方向に動くとき固定されます。

pegOffsetValue と組み合わせて注文の stopPx を設定するために使用します。ペグは execInst で指定されたトリガー価格（デフォルトは 'MarkPrice'）に設定されます。ストップセル注文とバイイフタッチ注文にはマイナスのオフセットを使用してください。

ordTypeが必要です: 'Stop'、'StopLimit'、'MarketIfTouched'、'LimitIfTouched'。

注文の追跡

注文 ID を自分で追跡したい場合は、注文ごとに一意の clOrdID を設定してください。この clOrdID は注文および関連する実行 (WebSocket 上のものも含む) にプロパティとして返ってきて、注文の取得またはキャンセルに使用できます。最大長は 36 文字です。

使用例:

```
// buy market order
BITMEX->REST_API->PlaceMarketOrder(bmosBuy, "XBTUSD", 100);
// sell limit order at 45000
BITMEX->REST_API->PlaceLimitOrder(bmosSell, "XBTUSD", 100, 45000.00);
// stop order at 48000
BITMEX->REST_API->PlaceStopOrder(bmosSell, "XBTUSD", 100, 48000.00);
```

API Bitfinex

Bitfinex

Bitfinex は世界最大かつ最先端の暗号通貨 取引プラットフォームの一つです。ユーザーはビットコイン、イーサリアム、リップル、EOS、ビットコイン キャッシュ、IOTA、NEO、ライトコイン、Ethereum Classicなどを交換できます。

Bitfinex WebSocket API バージョンは 2.0 です

Bitfinex の WebSocket チャンネルを介して送受信される各メッセージは JSON 形式でエンコードされています。

シンボルは取引ペアまたはマージン通貨にすることができます:

- 取引ペアシンボルは、ペアの前に "t" を付けて形成されます (例: tBTCUSD、tETHUSD)。
- マージン通貨のシンボルは、通貨の前に "f" を付けて形成されます (例: fUSD、fBTC、...)

接続が成功すると、**OnBitfinexConnect** イベントが発生し、パラメータとして Bitfinex API バージョン番号が取得されます。

Ping メソッドを呼び出してサーバーへの接続をテストできます。

サーバーから何らかの情報が送信された場合、**OnBitfinexInfoMessage** イベントで処理できます。このイベントでは、サーバーから送信されたメッセージに関する情報とともに Code と Message がパラメータとして渡されます。コードの例:

```
20051 : WebSocket サーバーの停止/再起動 (再接続してください)
20060 : メンテナンスモードに入ります。アクティビティを一時停止し、情報メッセージ 20061 を受信した後に再開してください (最大 120 秒かかります)。
20061 : メンテナンス終了。通常の活動を再開できます。すべてのチャンネルをサブスクライブ解除/再サブスクライブすることをお勧めします。
```

エラーが発生した場合、**OnBitfinexError** が発生し、エラーに関する情報が提供されます。エラーコードの例:

```
10000 : Unknown event
10001: 不明なペア
```

設定を変更するには、**Configuration** メソッドを呼び出して次のフラグのいずれかをパラメータとして渡してください:

```
CS_DEC_S = 8; // すべての小数を文字列として有効にする。
CS_TIME_S = 32; // すべての時間を日付文字列として有効にします。
CS_SEQ_ALL = 65536; // シーケンシングを有効にする ベータ機能
CHECKSUM = 131072; // 注文書の各反復でチェックサムを有効にします。書籍の各サイドのトップ 25 エントリを確認します。チェックサムは符号付き整数です。
```

パブリックチャンネルのサブスクライブ

認証なしでアクセスできるパブリックチャンネルがあります。すべてのメッセージは **OnBitfinexUpdate** イベントで発生します。

SubscribeTicker

ティッカーはマーケットの状態の高レベルな概要です。現在の最良の買い気配値と売り気配値、および最後の取引価格が表示されます。また、日次出来高や過去 1 日間の価格変動などの情報も含まれています。

```
// 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

このチャンネルは、Bitfinexで取引が発生するたびにトレードメッセージを送信します。価格、サイズ、時刻などの取引の重要な詳細がすべて含まれます。

```
// 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チャンネルを使用すると、Bitfinexのオーダーブックの状態を追跡できます。これは価格集約ベースで提供され、精度をカスタマイズできます。応答を受信すると、ブックのスナップショットを受信し、その後ブックへの変更があるたびに更新を受信します。

```
// on trading pairs (ex. tBTCUSD)
[
  CHANNEL_ID,
  [
    [
      PRICE,
      COUNT,
      AMOUNT
    ],
    ...
  ]
]

// on funding currencies (ex. fUSD)
[
  CHANNEL_ID,
  [
    [
      RATE,
      PERIOD,
      COUNT,
      AMOUNT
    ],
    ...
  ]
]
```

SubscribeRawOrderBook

これらは最も細粒度の板情報です。

```
// on trading pairs (ex. tBTCUSD)
[
  CHANNEL_ID,
  [
    [
      ORDER_ID,
      PRICE,
      AMOUNT
    ],
    ...
  ]
]

// on funding currencies (ex. fUSD)
[
  CHANNEL_ID,
  [
    [
      OFFER_ID,
      PERIOD,
      RATE,
      AMOUNT
    ],
    ...
  ]
]
```

SubscribeCandles

チャートのローソク足情報にアクセスする方法を提供します。時間枠:

1m : 1分
5m : 5分
15m : 15分
30m : 30 分
1h : one hour
3h : 3 時間
6h: 6 時間
12時間 : 12時間
1D : 1日
7D : 1週間
14D : 2週間
1M : 1ヶ月

```
[
  CHANNEL_ID,
  [
    [
      MTS,
      OPEN,
      CLOSE,
      HIGH,
      LOW,
      VOLUME
    ],
    ...
  ]
]
```

認証済みチャンネルをサブスクライブする

このチャンネルを使用すると、アカウントの状態を常に最新の状態に 保てます。ポジション、残高、注文、約定に関する更新を受信 できます。

サーバーに対して認証を行い、必要なパラメーターを設定するには、**Authenticate** メソッドを使用してください。

認証が完了すると、次の更新を受信します: 注文、ポジション、約定、ファンディングオファー、ファンディングクレジット、ファンディング ローン、ウォレット、残高情報、証拠金情報、ファンディング情報、ファンディング約定など。

サーバーからログオフしたい場合は **UnAuthenticate** メソッドをリクエストできます。

API Kucoin

Kucoin

Kucoinは国際的な多言語の暗号資産取引所です。KucoinのデータにアクセスするためのいくつかのAPIを提供しています。次のAPIがサポートされています。

1. **WebSocketストリーム**: 一部のメソッドを購読してリアルタイムでデータを取得できます。イベントはサーバーから購読者であるクライアントにプッシュされます。プロトコルとしてWebSocketを使用します。
2. **REST API**: クライアントはサーバーに市場データやアカウントデータをリクエストできます。認証にはAPIキー、シークレット、パスフレーズが必要で、プロトコルとしてHTTPsを使用します。

プロパティ

Kucoin API には2種類のメソッドがあります：公開と非公開。公開メソッドは認証なしでアクセスでき、例として取引価格の取得があります。ユーザーデータに関する非公開メソッドには Kucoin API キーの使用が必要です。

- **ApiKey**: kucoin アカウントで新しいAPI キーをリクエストし、その値をこのプロパティにコピーしてください。
- **ApiSecret**: API シークレットは REST_API のみ必要です。WebSocket API は一部のメソッドのみ ApiKey が必要です。
- **Passphrase**: Kucoinサーバーに接続するために必要な文字列。
- **Sandbox**: 有効にすると Kucoin デモアカウントに接続します（デフォルトは false）。
 - **HTTPLogOptions**: HTTP リクエストのログをテキストファイルに保存します。
 - **Enabled**: 有効にすると、WebSocket API のすべての HTTP リクエストが保存されます。
 - **FileName**: ログが保存されるファイルのフルパス。
 - **REST**: REST API リクエストのログをテキストファイルに保存します。
 - **Enabled**: 有効にすると、REST API のすべての HTTP リクエストが保存されます。
 - **FileName**: ログが保存されるファイルのフルパス。

最も一般的な使用方法

- **WebSockets API**
 - [WebSocket APIへの接続方法](#)
 - [WebSocketチャンネルのサブスクライブ方法](#)
- **REST API**
 - [マーケットデータの取得方法](#)
 - [プライベート REST API の使用方法](#)
 - [スポット取引の方法](#)
 - [プライベートリクエストの時刻](#)

WebSocket フィード

特定のサーバーからチャンネルメッセージにサブスクライブするには、クライアントサイドからサーバーにサブスクリプションメッセージを送信する必要があります。

サブスクリプションが成功すると、システムはあなたに ack メッセージを送信します。レスポンスが true に設定されているとき。

```
{
  "id": "1545910660739",
  "type": "ack"
}
```

トピックメッセージが生成されている間、システムはクライアント側に対応するメッセージを送信します。

以下のサブスクリプション/サブスクリプション解除メソッドがサポートされています。

パブリックチャンネル

メソッド	パラメータ	説明
SubscribeSymbolTicker	銘柄	BBO の変更のプッシュを受け取るためにこのトピックをサブスクライブします。1秒以内に変更がない場合、プッシュされません。最新の BBO を100ms ごとにプッシュします。前回のデータから変更がない場合、プッシュされません。
SubscribeAllSymbolsTicker		すべてのマーケットシンボルの BBO 変更のプッシュを受け取るためにこのトピックをサブスクライブしてください。
SubscribeSymbolSnapshot	銘柄	単一のシンボルのスナップショットデータを取得するためにサブスクライブします。スナップショットデータは2秒間隔でプッシュされます。
SubscribeMarketSnapshot	Market	このトピックをサブスクライブして、市場全体のスナップショットデータを取得します。スナップショットデータは2秒間隔でプッシュされます。
SubscribeLevel2MarketData	銘柄	このトピックをサブスクライブすると Level2 の板情報データを取得できます。WebSocket サブスクリプションが成功すると、WebSocket 経由でプッシュされる増分変更データが送信されます。
SubscribeLevel2_5BestAskBid	銘柄	システムは5件の最良の売値/買値注文データを返します。これは100ミリ秒ごとのスナップショットデータです（つまり、5件の最良の売値/買値注文データが100ミリ秒ごとにリアルタイムで返されます）。
SubscribeLevel2_50BestAskBid	銘柄	システムは50件のベスト Ask/Bid 注文データを返します。これは100ミリ秒ごとのスナップショットデータです（つまり、50件のベスト Ask/Bid 注文データがリアルタイムで100ミリ秒ごとに返されます）。
SubscribeKlines	銘柄	K ラインデータを取得するにはこのトピックにサブスクライブしてください。
SubscribeMatchExecutionData	銘柄	このトピックにサブスクライブすると、Level 3 の一致イベントデータフローを取得します。約定された注文ごとに、システムは以下の形式で一致メッセージを送信します。
SubscribeIndexPrice	銘柄	このトピックにサブスクライブしてマージン取引のインデックス価格を取得します。
SubscribeMarkPrice	銘柄	信用取引のマーク価格を取得するには、このトピックをサブスクライブしてください。
SubscribeOrderBookChanged	銘柄	信用取引のオーダーブックの変化を取得するには、このトピックをサブスクライブしてください。
SubscribeLevel1	銘柄	シンボルのレベル 1 最良気配値データをサブスクライブします。

ACK パラメータが True に設定されている場合、サブスクリプション/サブスクリプション解除が成功した後、クライアントはその旨のメッセージを受信します。

プライベートチャンネル

Kucoin アカウントから取得した有効な ApiKey が必要です。ApiKey、ApiSecret、Passphrase はクライアント API コンポーネントの Kucoin プロパティに設定する必要があります。

変化があるたびに次のデータがクライアントにプッシュされます。任意のメソッドをサブスクライブする必要はありません。有効な ApiKey が設定されていれば自動的に行われます。

メソッド	説明
SubscribeTradeOrders	このトピックはお客様の注文のすべての変更イベントをプッシュします。
SubscribeAccountBalance	アカウント残高が変化したとき、このメッセージを受信します。メッセージには変化の詳細が含まれます。
SubscribePositionStatus	ポジションステータスが変化すると、システムは変更イベントをプッシュします。
SubscribeMarginTradeOrders	注文がオーダーブックに入ると、システムはこのメッセージを貸し手にプッシュします。
SubscribeStopOrder	システムがストップ注文を受け取ると、「open」タイプのメッセージを受信します。これは、この注文がシステムに入力され、トリガーされるのを待機していることを意味します。
SubscribeTradeOrdersV2	強化された注文更新通知のためにトレード注文V2チャンネルにサブスクライブします。
SubscribeCrossMarginPosition	クロスマージンポジションの更新にサブスクライブします。クロスマージンポジションが変化すると、システムが変更イベントをプッシュします。
SubscribeIsolatedMarginPosition	隔離マージンポジションの更新をサブスクライブします。隔離マージンポジションが変更されると、システムは変更イベントをプッシュします。

REST API

すべてのエンドポイントは JSON オブジェクトまたは配列を返します。

パブリック API エンドポイント

これらのエンドポイントには認証なしでアクセスできます。

汎用エンドポイント

メソッド	パラメータ	説明
GetServiceStatus		Rest API への接続を確認し、サービスステータスを取得します。
GetServerTime		REST API への接続をテストして現在のサーバー時刻を取得します。

Market Data エンドポイント

メソッド	パラメータ	説明
GetSymbolList	Market	取引可能な通貨ペアの一覧を取得するには、このエンドポイントにリクエストします。取引シンボルの市場情報を取得する場合は
GetTicker	銘柄	このエンドポイントにリクエストしてレベル 1 のマーケットデータを取得します。返される値には、最良の買い値と数量、最良の売り値と数量、最終取引価格と最終取引数量が含まれます。
GetAllTickers		マーケット内のすべての取引ペアのマーケットティッカーをリクエストします（24 時間取引量を含む）。

Get24hrStats	銘柄	このエンドポイントにリクエストして、過去 24 時間の指定されたティッカーの統計を取得してください。
GetMarketList		このエンドポイントを介してリクエストし、取引市場全体のトランザクション通貨を取得します。
GetPartOrderBook20	銘柄	このエンドポイントを介してリクエストし、シンボルのオープン注文のリストを取得します。レベル2のオーダーブックにはすべての買い気配と売り気配が含まれ（価格ごとに集約）、このレベルではアクティブな各価格に対して1つのサイズのみを返します（その価格に対して単一の注文しかないかのように）。システムはオーダーブックの20件のデータ（売り気配と買い気配のデータ）を返します。
GetPartOrderBook100	銘柄	このエンドポイントにリクエストして、シンボルのオープン注文リストを取得します。レベル2注文書にはすべての買い注文と売り注文が含まれます（価格ごとに集約）。このレベルでは、各アクティブな価格に対して1つのサイズのみが返されます（その価格に1つの注文しかない場合と同様）。システムは注文書のデータ（買い注文と売り注文のデータ）を100件返します。
GetFullOrderBook	銘柄	このエンドポイントを通じてリクエストして、指定されたシンボルのオーダーブックを取得します。レベル2オーダーブックはすべての買い注文と売り注文を含みます（価格で集計）。このレベルは各価格に対して1つの集計サイズのみを返します（その価格に対して1つの注文のみがあるかのように）。このAPIはフルデプスのデータを返します。
GetKLines	銘柄	指定されたシンボルの Kライン（ローソク足）を取得するためにこのエンドポイントにリクエストしてください。データはリクエストされたタイプに基づいてグループ化されたバケットで返されます。
GetCurrencies		このエンドポイントを使用して通貨リストを取得します。
GetCurrencyDetail	通貨	このエンドポイントにリクエストして指定された通貨の通貨詳細を取得します。
GetFiatPrice		このエンドポイントにリクエストして指定された通貨の通貨詳細を取得します。
GetPartOrderBook1	銘柄	このエンドポイントにリクエストして、シンボルのレベル1最良気配値（最良売買気配）を取得します。

プライベート API エンドポイント

サーバーによる認証を取得するために APIKey と APISecret が必要です。

User EndPoints

メソッド	パラメータ	説明
GetAllSubAccounts		このインターフェースを通じてすべてのサブユーザーのユーザー情報を取得できます。
GetListAccounts		アカウントの一覧を取得します。

GetAccount	AccountId	単一アカウントの情報。accountIdがわかっている場合にこのエンドポイントを使用してください。
GetAccountBalanceSubAccount	SubUserId	このエンドポイントは subUserId で指定されたサブユーザーのアカウント情報を返します。
InnerTransfer		この API エンドポイントは、アカウント間で内部的に資金を移動するために使用できます。ユーザーはメインアカウント、取引アカウント、クロスマージンアカウント、および分離マージンアカウント間で無料で資金を移動できます。メインアカウント、クロスマージンアカウント、および取引アカウントから先物アカウントへの資金移動はサポートされていますが、先物アカウントから他のアカウントへの資金移動はサポートされていません。
GetDepositAddresses	通貨	通貨のデポジットアドレスを取得します。
CreateDepositAddress	通貨	通貨の新しい入金アドレスを作成します。
GetDepositList		入金履歴を取得します。
GetAccountLedgers		アカウント台帳のエントリを取得します。
GetTradeFees	シンボル	指定したシンボルの取引手数料を取得します。

Withdraw EndPoints

メソッド	パラメータ	説明
GetWithdrawalsList		出金のリストを取得します。
GetHistoricalWithdrawalsList		KuCoin V1 の過去の出金一覧です。
GetWithdrawalsQuotas	通貨	出金クォータを取得
ApplyWithdraw	Currency, Address, Amount	Withdrawを作成する
CancelWithdraw	WithdrawalId	キャンセルできるのは、PROCESSINGステータスの出金要求のみです。

取引エンドポイント

メソッド	パラメータ	説明
PlaceOrder		2種類の注文を発注できます: 指値と成行。注文はアカウントに十分な資金がある場合にのみ発注できます。注文が発注されると、注文期間中アカウントの資金が保留されます。保留される資金の量と種類は、指定された注文タイプとパラメータによって異なります。
PlaceMarketOrder		成行注文を発注します。
PlaceLimitOrder		指値注文を発注します。
PlaceMarginOrder		マージン注文を発注します。
CancelOrder		注文 ID で注文をキャンセルします。
CancelOrderByClientOid		クライアント注文 ID によって注文をキャンセルします。
CancelAllOrders		すべての未決注文をキャンセルします。

ListOrders		現在の注文リストを取得するにはこのエンドポイントにリクエストしてください。アイテムはページ分割され、最新のものが最初に表示されるようにソートされています。
GetRecentOrders		このエンドポイントを通じて過去 24 時間の 1000 件の注文を取得するリクエストを行います。
GetOrder		このエンドポイントを介してリクエストし、order IDで単一の注文情報を取得します。
GetOrderByClientOid		このエンドポイントにリクエストして、クライアント注文 ID で単一の注文情報を取得してください。
ListFills		最近の約定を取得するにはこのエンドポイントにリクエストしてください。
GetRecentFills		このエンドポイントを介してリクエストし、過去24時間の最大1000件の約定のリストを取得します。
PlaceStopOrder		ストップ注文を発注します。
PlaceStopMarketOrder		ストップマーケット注文を発注します。
PlaceStopLimitOrder		ストップリミット注文を発注します。
CancelStopOrder		注文 ID でオープンストップ注文をキャンセルします。
CancelStopOrderByClientOid		クライアント注文 ID でオープンストップ注文をキャンセルします。
CancelAllStopOrders		すべてのストップ注文をキャンセル
GetStopOrder		注文 ID からストップ注文情報を取得するにはこのインターフェースにリクエストします。
GetStopOrderByClientOid		このインターフェースを介してリクエストし、Client order IDでストップ注文情報を取得します。
ListStopOrders		このエンドポイントを介して要求すると、現在のトリガーされていないストップ注文の一覧を取得します。項目はページ分割され、最新のものが先頭に表示されるようにソートされます。
PlaceHFOOrder		高頻度注文を発注します。
CancelHFOOrder		注文 ID で高頻度注文をキャンセルします。
CancelHFOOrderByClientOid		クライアント注文 ID で高頻度注文をキャンセルします。
CancelAllHFOOrders		すべての高頻度注文をキャンセルします。
GetHFActiveOrders		アクティブな高頻度注文を取得します。
GetHFDoneOrders		完了した高頻度注文を取得します。
GetHFOOrder		注文 ID で特定の高頻度注文を取得します。

イベント

Kucoin のメッセージは TsgcWebSocketClient コンポーネントで受信されます。以下のイベントを使用できます：

OnConnect

Kucoinサーバーへの接続が成功した後。

OnDisconnect

Kucoin サーバーからの切断後

OnMessage

サーバーからクライアントに送信されたメッセージは、このイベントで処理されます。

OnError

プロトコルにエラーがある場合、このイベントが呼び出されます。

OnException

未処理の例外がある場合、このイベントが呼び出されます。

さらに、Kucoin API コンポーネントには **OnKucoinHTTPException** という特定のイベントがあり、HTTP リクエスト（REST API または WebSocket フィード）の呼び出しでエラーが発生するたびに発生します。

Kucoin | WebSocket API への接続

Kucoin WebSocket APIに接続するには、新しいKucoin APIクライアントを作成してTsgcWebSocketClientにアタッチするだけです。
以下の例をご覧ください:

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_Kucoin *oKucoin = new TsgcWSAPI_Kucoin();
oKucoin->Client = oClient;
oClient->Active = true;
```

Kucoin | WebSocket チャンネルのサブスクライブ

Kucoin では、マーケットデータや注文などのリアルタイム更新を取得するためにサブスクライブできるさまざまなチャンネルを提供しています。以下に Ticker のサブスクライブ方法のサンプルを示します：

```
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 | マーケットデータの取得

Kucoin は REST エンドポイントを通じて公開のマーケットデータを提供しています。これらのエンドポイントのいずれかを呼び出すと、リクエストされたマーケットデータのスナップショットが取得されます。マーケットデータエンドポイントは認証を必要としないため、すべてのユーザーが自由に利用できます。

使用例: ティッカー BTC-USDT のスナップショットを取得するには、次の呼び出しを実行します。

```
TsgcWSAPI_Kucoin *oKucoin = new TsgcWSAPI_Kucoin(this);
ShowMessage(oKucoin->REST_API->GetTicker("
BTC-USDT
"));
```

Kucoin | プライベート REST API

Kucoin REST API はパブリックとプライベートのエンドポイントを提供しています。プライベートエンドポイントではトランザクションのセキュリティを高めるためにメッセージへの署名が必要です。まず、Kucoinアカウントにログインして新しいAPIを作成する必要があります。次の値が得られます。

- ApiKey
- ApiSecret
- パスフレーズ

これらのフィールドは Kucoin API クライアントコンポーネントの Kucoin プロパティで設定する必要があります。

設定が完了したら、Kucoin Pro REST API にプライベートリクエストを送信できます。

*プライベートリクエストには、ローカルマシンのローカル時刻が同期されている必要があります。同期されていない場合、リクエストは Kucoin サーバーに拒否されます。これについては次の記事を参照してください: [Kucoin プライベートリクエスト時刻](#)。

```
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 | スポット取引

Kucoin では REST API を使用してスポット取引が可能です。

設定

まず Kucoin アカウントで **API キー**を作成し、Spot 取引の権限を追加する必要があります。

これが完了すると、現物取引を開始できます。

まず、**ApiKey**、**ApiSecret**、**Passphrase**を設定します。Kucoin Clientコンポーネントで設定し、これはKucoin サーバーに送信されるリクエストの署名に使用されます。

注文を発注する

新しい注文を出すには、Kucoin クライアントコンポーネントの **REST_API.PlaceOrder** メソッドを呼び出してください。

注文のタイプ(market、limitなど)に応じて、APIは多かれ少なかれ フィールドを必要とします。

パラメータ

パラメータ	type	説明
clientOid	String	ユーザーが注文を識別するために作成する一意の注文 ID（例：UUID）。
side	String	買い または 売り
symbol	String	有効な取引シンボルコード。例: ETH-BTC
type	String	[オプション] limit または market （デフォルトは limit ）
remark	String	[オプション] 注文の備考。長さはUTF-8で100文字を超えることはできません。
stp	String	[オプション] セルフトレード防止、 CN 、 CO 、 CB または DC
tradeType	String	[オプション] 取引の種類: TRADE （スポット取引）、 MARGIN_TRADE （マージン取引）。デフォルトは TRADE 。注意: システムのパフォーマンスを向上させ、注文の発注と処理を高速化するために、KuCoin はマージン向けの新しい注文発注インターフェースを追加しました。現在のインターフェースを使用しているトレーダーは、できるだけ早く新しいインターフェースに移行してください。現在のインターフェースは 2021 年 5 月 1 日（UTC）以降、マージン注文を受け付けなくなりま。その際、KuCoin はアナウンスでユーザーに通知しますのでご注意ください。

指値注文パラメータ

パラメータ	type	説明
price	String	基準通貨あたりの価格
size	String	購入または売却するベース通貨の数量
timeInForce	String	[オプション] GTC 、 GTT 、 IOC 、または FOK （デフォルトは GTC ）。 有効期間 を参照してください。

パラメータ	type	説明
cancelAfter	long	[オプション] 次の秒数経過後にキャンセルします n 秒。 timeInForce が GTT である必要があります
postOnly	boolean	[オプション] ポストオンリーフラグ。 timeInForce が IOC または FOK の場合は無効。
hidden	boolean	[オプション] 注文はオーダーブックに表示されません
iceberg	boolean	[オプション] 注文の一部のみが板に表示されます。
visibleSize	String	[Optional] iceberg注文の最大可視サイズ

MARKET ORDER PARAMETERS

パラメータ	type	説明
size	String	[オプション] 基準通貨での希望数量
funds	String	[オプション] 使用したい見積もり通貨の希望数量

注文を送信する際、2つの可能性があります。

1. **成功:** PlaceOrder関数はKucoinサーバーから送信されたメッセージを返します。
2. **Error:** 例外は OnKucoinHttpException イベントで返されます。

1 BTC-USDT 成行注文を発注する

```
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
));
```

40000 で 1 BTC-USDT の指値注文を出す

```
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 | プライベートリクエスト時刻

Kucoin にプライベートリクエストを行うと、リクエストのセキュリティを高めるためにメッセージが署名されます。メッセージにはローカル時刻が含まれて署名済みメッセージの中に送信されますが、ローカル時刻と Kucoin サーバーとの差が 5 秒を超えるとリクエストは拒否されます。ローカル時刻が同期していることを確認することが重要です。OS の時刻同期メソッドを使用して確認できます。

GetServerTime メソッドを呼び出すことで Kucoin サーバーの時刻を確認できます。これは Kucoin サーバーの時刻を返します。

API Kucoin 先物

Kucoin Futures

Kucoinは国際的な多言語の暗号資産取引所です。KucoinのデータにアクセスするためのいくつかのAPIを提供しています。次のAPIがサポートされています。

1. **WebSocketストリーム**: 一部のメソッドを購読してリアルタイムでデータを取得できます。イベントはサーバーから購読者であるクライアントにプッシュされます。プロトコルとしてWebSocketを使用します。
2. **REST API**: クライアントはサーバーに市場データやアカウントデータをリクエストできます。認証にはAPIキー、シークレット、パスフレーズが必要で、プロトコルとしてHTTPsを使用します。

プロパティ

Kucoin API には2種類のメソッドがあります：パブリックとプライベートです。パブリックメソッドは認証なしでアクセスできます（例：ティックャー価格の取得）。プライベートおよびユーザーデータに関連するメソッドはKucoin API キーの使用が必要です。

- **ApiKey**: kucoin アカウントで新しいAPI キーをリクエストし、その値をこのプロパティにコピーしてください。
- **ApiSecret**: API シークレットは REST_API のみ必要です。WebSocket API は一部のメソッドのみ ApiKey が必要です。
- **Passphrase**: Kucoinサーバーに接続するために必要な文字列。
- **Sandbox**: 有効にすると Kucoin デモアカウントに接続します（デフォルトは false）。
 - **HTTPLogOptions**: HTTP リクエストのログをテキストファイルに保存します。
 - **Enabled**: 有効にすると、WebSocket API のすべての HTTP リクエストが保存されます。
 - **FileName**: ログが保存されるファイルのフルパス。
 - **REST**: REST API リクエストのログをテキストファイルに保存します。
 - **Enabled**: 有効にすると、REST API のすべての HTTP リクエストが保存されます。
 - **FileName**: ログが保存されるファイルのフルパス。

最も一般的な使用方法

- **WebSocket API**
 - [WebSocket API への接続方法](#)
 - [WebSocket チャンネルのサブスクライブ方法](#)
- **REST API**
 - [マーケットデータの取得方法](#)
 - [プライベートREST APIの使用方法](#)
 - [先物取引の方法](#)
 - [Private Requests Time](#)

WebSocket フィード

特定のサーバーからチャンネルメッセージにサブスクライブするには、クライアントサイドからサーバーにサブスクリプションメッセージを送信する必要があります。サブスクリプションが成功すると、システムはあなたに ack メッセージを送信します。レスポンスが true に設定されているとき。

```
{
  "id": "1545910660739",
  "type": "ack"
}
```

トピックメッセージが生成されている間、システムはクライアント側に対応するメッセージを送信します。

以下のサブスクリプション/サブスクリプション解除メソッドがサポートされています。

パブリックチャンネル

メソッド	パラメータ	説明
SubscribeSymbolTickerV2	銘柄	このトピックをサブスクライブすると、BBO の変更がリアルタイムでプッシュされます。サブスクリプション後、板に変更があると、システムはリアルタイムのティッカー シンボル情報をプッシュします。タイムリーな情報のために新しいトピックの使用を推奨します。
SubscribeSymbolTicker	銘柄	このトピックをサブスクライブして BBO 変更のリアルタイムプッシュを取得します。ティッカーチャンネルはマッチが発生するたびにリアルタイムの価格更新を提供します。複数の注文が同時にマッチされた場合、最後のマッチングイベントのみがプッシュされます。
SubscribeLevel2MarketData	銘柄	レベル2のオーダーブックデータを取得するには、このトピックをサブスクライブします。
SubscribeExecutionData	銘柄	実行された各注文について、システムは以下の形式でマッチメッセージを送信します。
SubscribeLevel2_5BestAskBid	銘柄	最大でも100ミリ秒ごとに返されます。
SubscribeLevel2_50BestAskBid	銘柄	最大でも100ミリ秒ごとに返されます。
SubscribeContractMarketData	銘柄	契約のマーケットデータを取得するためにこのトピックにサブスクライブします。
SubscribeSystemAnnouncements	銘柄	このトピックを購読してシステムのお知らせを受け取ります。
SubscribeTransactionStatistics	銘柄	トランザクション統計は5秒ごとにユーザーにプッシュされます。
SubscribeKlines	銘柄	契約のKライン（ローソク足）データをサブスクライブします。
SubscribeFundingFeeSettlement	銘柄	ファンディング手数料の精算通知にサブスクライブします。

ACK パラメータが True に設定されている場合、サブスクリプション/サブスクリプション解除が成功した後、クライアントはその旨のメッセージを受信します。

プライベートチャンネル

Kucoin アカウントから取得した有効な ApiKey が必要です。ApiKey、ApiSecret、Passphrase はクライアント API コンポーネントの Kucoin プロパティに設定する必要があります。

変化があるたびに次のデータがクライアントにプッシュされます。任意のメソッドをサブスクライブする必要はありません。有効な ApiKey が設定されていれば自動的に行われます。

メソッド	説明
SubscribeTradeOrders	このトピックはお客様の注文のすべての変更イベントをプッシュします。
SubscribeAccountBalance	アカウント残高が変化したとき、このメッセージを受信します。メッセージには変化の詳細が含まれます。
SubscribePositionChange	ポジションステータスが変化すると、システムは変更イベントをプッシュします。

SubscribeStopOrder	システムがストップ注文を受け取ると、「open」タイプのメッセージを受信します。これは、この注文がシステムに入力され、トリガーされるのを待機していることを意味します。
SubscribeMarginMode	マージンモードの変更をサブスクライブします。マージンモードが更新されると、システムが変更イベントをプッシュします。
SubscribeCrossMarginLeverage	クロスマージンのレバレッジ変更をサブスクライブします。クロスマージンのレバレッジが更新されると、システムは変更イベントをプッシュします。

REST API

すべてのエンドポイントは JSON オブジェクトまたは配列を返します。

パブリック API エンドポイント

これらのエンドポイントには認証なしでアクセスできます。

汎用エンドポイント

メソッド	パラメータ	説明
GetServiceStatus		Rest API への接続を確認し、サービスステータスを取得します。
GetServerTime		REST API への接続をテストして現在のサーバー時刻を取得します。

Market Data エンドポイント

メソッド	パラメータ	説明
GetOpenContractList		すべてのオープンコントラクトの情報を取得するリクエストを送信します。
GetOrderInfoContract		指定されたコントラクトの情報を取得するリクエストを送信します。
GetTicker	銘柄	リアルタイムティッカーには、最終取引価格、最終取引サイズ、トランザクション ID、流動性テイク側、最良の買い値と数量、最良の売り値と数量、および注文のトランザクション時刻が含まれます。これらのメッセージは Websocket でも取得できます。シーケンス番号は Websocket によってプッシュされたメッセージが継続しているかどうかを判断するために使用されます。
GetPartOrderBook20	銘柄	シンボルの集約オープン注文のスナップショットを取得します。
GetPartOrderBook100	銘柄	シンボルの集約オープン注文のスナップショットを取得します。
GetFullOrderBook	銘柄	シンボルの集約オープン注文のスナップショットを取得します。
GetLevel2PullingMessages	銘柄	Websocketによってプッシュされるメッセージが連続していない場合は、次のリクエストを送信してデータを再取得し、シーケンスが欠落していないことを確認

		できます。リクエストにおいて、startパラメータは最後に受信したメッセージのシーケンス番号に1を加えたものであり、endパラメータは現在受信したメッセージのシーケンス番号から1を引いたものです。メッセージを再取得してローカルの取引所オーダーブックに適用した後、Websocketのインクリメンタルフィードを介してオーダーブックの更新を続けることができます。endパラメータとstartパラメータの差が500を超える場合は、このリクエストの使用を停止し、レベル2オーダーブックを再構築することをお勧めします。
GetTradeHistory	銘柄	銘柄の最新 100 件の取引を一覧表示します。
GetInterestRateList	銘柄	金利リストを確認します。
GetIndexList	銘柄	インデックスリストを確認します。
GetCurrentMarkPrice	銘柄	現在のマーク価格を確認します。
GetPremiumIndex	銘柄	プレミアムインデックスを取得するリクエストを送信します。
GetCurrentFundingRate	銘柄	現在のマーク価格を確認するリクエストを送信します。
GetKLine	銘柄	コントラクトの K ラインデータを取得します

プライベート API エンドポイント

サーバーによる認証を取得するために APIKey と APISecret が必要です。

User EndPoints

メソッド	パラメータ	説明
GetAccountOverview		Get Account Overview
GetTransactionHistory		未決済ポジションがある場合、返される最初のページのステータスは Pending となり、現在の 8 時間決済期間の実現損益を示します。現在のページの最小オフセット番号を offset フィールドに指定してページを切り替えてください。

取引エンドポイント

メソッド	パラメータ	説明
PlaceOrder		注文には指値と成行の2種類があります。注文は口座に十分な資金がある場合にのみ発注できます。注文が発注されると、注文期間中は資金が保留されます。保留される資金の量は注文の種類と指定されたパラメータによって異なります。
PlaceMarketOrder		成行注文を発注します。
PlaceLimitOrder		指値注文を発注します。
CancelOrder		注文 ID で注文をキャンセルします。

LimitOrderMassCancellation		すべてのオープン注文をキャンセルします（ストップ注文を除く）。応答は、キャンセルされた注文の orderID のリストです。
StopOrderMassCancellation		未トリガーのストップ注文をすべてキャンセルします。レスポンスはキャンセルされたストップ注文の orderID のリストです。トリガー済みのストップ注文をキャンセルするには、「Limit Order Mass Cancellation」を使用してください。
GetOrderList		現在の注文を一覧表示します。
GetUntriggeredStopOrderList		トリガーされていないストップ注文リストを取得します。
GetListOrdersCompleted24hr		直近24時間における直近1000件の注文のリストを取得します。低レイテンシで直近の取引済み注文履歴を取得する必要がある場合は、このエンドポイントにクエリできます。
GetOrder		注文 ID で単一の注文を取得します（逆指値注文を含む）。
GetOrderByClientOid		クライアント注文 ID で単一の注文を取得します（ストップ注文を含む）。
GetFills		最近の約定のリストを取得します。
GetRecentFills		過去24時間の最近1000件の約定履歴を取得します。低遅延で最近の取引注文履歴を取得する必要がある場合は、このエンドポイントを使用してください。
ActiveOrderValueCalculation		このエンドポイントを照会して、すべてのアクティブな注文の合計数と総額を取得できます。
GetPositionDetails		指定したポジションの詳細情報を取得します。
GetPositionList		指定したポジションの詳細情報を取得します。
AutoDepositMargin		自動証拠金入金の有効化/無効化
AddMarginManually		マージンを手動で追加
ObtainFuturesRiskLimitLevel		このインターフェースを使用して特定の契約のリスク限度レベルに関する情報を取得できます
AdjustRiskLimitLevel		このインターフェースはリスク制限レベルの調整用です。レベルを調整するとオープンオーダーがキャンセルされます。レスポンスは調整リクエストの送信が成功したかどうかのみを示します。
GetFundingHistory		資金調達履歴の取得リクエストを送信します。
GetMaxOpenSize		コントラクトの最大建玉サイズを取得します。
SwitchMarginMode		cross marginモードとisolated marginモードを切り替えます。
GetMarginMode		契約の現在のマージンモードを取得します。

イベント

Kucoin のメッセージは TsgcWebSocketClient コンポーネントで受信されます。以下のイベントを使用できます：

OnConnect

Kucoinサーバーへの接続が成功した後。

OnDisconnect

Kucoin サーバーからの切断後

OnMessage

サーバーからクライアントに送信されたメッセージは、このイベントで処理されます。

OnError

プロトコルにエラーがある場合、このイベントが呼び出されます。

OnException

未処理の例外がある場合、このイベントが呼び出されます。

さらに、Kucoin API コンポーネントには **OnKucoinHTTPException** という特定のイベントがあり、HTTP リクエスト（REST API または WebSocket フィード）の呼び出しでエラーが発生するたびに発生します。

Kucoin | Futures WebSocket API への接続

Kucoin WebSocket APIに接続するには、新しいKucoin APIクライアントを作成してTsgcWebSocketClientにアタッチするだけです。
以下の例をご覧ください:

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();  
TsgcWSAPI_Kucoin_Futures  
*oKucoin = new  
TsgcWSAPI_Kucoin_Futures  
(  
);  
oKucoin->Client = oClient;  
oClient->Active = true;
```

Kucoin | Futures WebSocket チャンネルのサブスクライブ

Kucoin では、マーケットデータや注文などのリアルタイム更新を取得するためにサブスクライブできるさまざまなチャンネルを提供しています。以下に Ticker のサブスクライブ方法のサンプルを示します：

```
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

Kucoin は REST エンドポイントを通じて公開のマーケットデータを提供しています。これらのエンドポイントのいずれかを呼び出すと、リクエストされたマーケットデータのスナップショットが取得されます。マーケットデータエンドポイントは認証を必要としないため、すべてのユーザーが自由に利用できます。

使用例: ティッカー BTC-USDT のスナップショットを取得するには、次の呼び出しを実行します。

```
TsgcWSAPI_Kucoin_Futures  
  
*oKucoin = new  
TsgcWSAPI_Kucoin_Futures  
(this);  
ShowMessage(oKucoin->REST_API->GetTicker("XBTUSDM  
"));
```

Kucoin | Futures Private REST API

Kucoin REST API はパブリックとプライベートのエンドポイントを提供しています。プライベートエンドポイントではトランザクションのセキュリティを高めるためにメッセージへの署名が必要です。まず、Kucoinアカウントにログインして新しいAPIを作成する必要があります。次の値が得られます。

- ApiKey
- ApiSecret
- パスフレーズ

これらのフィールドは Kucoin API クライアントコンポーネントの Kucoin プロパティで設定する必要があります。

設定が完了したら、Kucoin Pro REST API にプライベートリクエストを送信できます。

*プライベートリクエストは、ローカルマシンの時刻が同期されている必要があります。同期されていない場合、リクエストは Kucoin サーバーに拒否されます。これについては次の記事を参照してください: [Kucoin プライベートリクエスト時刻](#)。

```
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 | 先物取引

Kucoin では REST API を使用して先物取引ができます。

設定

まず、Kucoinアカウントで **APIキー** を作成し、先物取引の権限を追加する必要があります。

完了したら、先物取引を開始できます。

まず、**ApiKey**、**ApiSecret**、**Passphrase**を設定します。Kucoin Clientコンポーネントで設定し、これはKucoinサーバーに送信されるリクエストの署名に使用されます。

注文を発注する

新しい注文を出すには、Kucoin クライアントコンポーネントの **REST_API.PlaceOrder** メソッドを呼び出してください。

注文のタイプ(market、limitなど)に応じて、APIは多かれ少なかれ フィールドを必要とします。

パラメータ

パラメータ	type	説明
clientOid	String	注文を識別するためにユーザーが作成した一意の注文ID（例: UUID）。数字、文字、アンダースコア（_）、セパレータ（-）のみ使用可能。
side	String	buy （買い）または sell （売り）
symbol	String	有効なコントラクトコード（例：XBTUSD）。M
type	String	[オプション] limit または market のいずれか
レバレッジ	String	注文のレバレッジ
remark	String	[オプション] 注文に対する備考。UTF-8 文字数で100文字を超えることはできません
stop	String	[オプション] down または up のいずれか。 stopPrice と stopPriceType の定義が必要です
stopPriceType	String	[オプション] TP 、 IP 、または MP のいずれか。 stop が指定されている場合は定義が必要です。
stopPrice	String	[オプション] stop が指定されている場合に定義が必要です。
reduceOnly	boolean	[オプション] ポジションサイズのみを縮小するためのマーク。デフォルトでは false に設定されています。reduceOnly が true の場合、ポジションサイズを設定する必要があります。
closeOrder	boolean	[オプション] ポジションをクローズするためのマーク。デフォルトは false です。closeOrder が true の場合、すべてのポジションをクローズします。
forceHold	boolean	[オプション] ポジションサイズを縮小する注文であっても、注文のために資金を強制的に保持するマーク。これにより、注文がオーダーブックに残り、ポジションサイズが変わったときにキャンセルされないようにします。デフォルトでは false に設定されています。

指値注文パラメータ

Param	type	説明
price	String	指値価格
size	Integer	注文サイズ。正の数でなければなりません。
timeInForce	String	[オプション] GTC 、 IOC （デフォルトは GTC ）、 Time In Force を読む
postOnly	boolean	[オプション] ポストオンリーフラグ。 timeInForce が IOC の場合は無効です。postOnly を選択した場合、hidden または iceberg は選択できません。
hidden	boolean	[オプション] 板に表示されない注文です。hidden を選択した場合、postOnly は選択できません。
iceberg	boolean	[オプション] 注文のうち表示される部分のみが板に表示されます。iceberg を選択した場合、postOnly は選択できません。
visibleSize	Integer	[オプション] アイスバーグ注文の最大表示サイズ

MARKET ORDER PARAMETERS

パラメータ	type	説明
size	Integer	[オプション] 買い付けまたは売り付ける契約数量

注文を送信する際、2つの可能性があります。

1. **成功:** PlaceOrder関数はKucoinサーバーから送信されたメッセージを返します。
2. **Error:** 例外は OnKucoinHttpException イベントで返されます。

マーケット注文 1 XBTUSD を発注

```

TsgcWSAPI_Kucoin_Futures
*oKucoin = new
TsgcWSAPI_Kucoin_Futures
(this);
oKucoin->Kucoin->ApiKey = "<api key>";
oKucoin->Kucoin->ApiSecret = "<api secret>";
oKucoin->Kucoin->Passphrase = "<passphrase>";
ShowMessage(oKucoin->REST_API->
PlaceMarketOrder
(
kosBuy, "XBTUSD", 1
));

```

40000で1 XBTUSDの指値注文を発注

```
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 | フューチャーズプライベートリクエスト時間

Kucoin にプライベートリクエストを行うと、リクエストのセキュリティを高めるためにメッセージが署名されます。メッセージにはローカル時刻が含まれて署名済みメッセージの中に送信されますが、ローカル時刻と Kucoin サーバーとの差が 5 秒を超えるとリクエストは拒否されます。ローカル時刻が同期していることを確認することが重要です。OS の時刻同期メソッドを使用して確認できます。

GetServerTime メソッドを呼び出すことで Kucoin サーバーの時刻を確認できます。これは Kucoin サーバーの時刻を返します。

API 3Commas

3Commas

サポートされている API

- **WebSockets API:** パブリックなWebSocketサーバーに接続し、リアルタイムのマーケットデータ更新を提供します。
- **REST API:** REST APIには、アカウントと注文管理 および公開市場データ用のエンドポイントがあります。

WebSockets API

WebSocket フィードはトレードとディールのリアルタイム市場データ更新を提供します。

次の**Public channels**にサブスクライブできます:

メソッド	Arguments	説明
SubscribeSmartTrades		
SubscribeDeals		

これらのチャンネルは 3Commas サーバーに対する**認証**が必要です。まず 3Commas アカウントで API キーをリクエストし、コンポーネントの ThreeComas プロパティに値を設定してください。

- ApiKey
- ApiSecret

サブスクリプションが成功した場合、**OnThreeCommasConfirmSubscription** イベントが呼び出されます。失敗した場合は **OnThreeCommasRejectSubscription** イベントが呼び出され、**aRawMessage** パラメータを使用して拒否の理由を取得できます。

REST API

接続テスト

メソッド	Arguments	説明
GetPing		
GetServerTime		サーバー時刻を返します

アカウント

メソッド	Arguments	説明
GetAccounts		ユーザーが接続している

		取引所のリスト
GetMarketList		サポートされているマーケットリスト
GetMarketPairs	aMarketCode: マーケットのコード	すべての市場ペア
GetCurrencyRatesWithLeverageData	aMarketCode: マーケットのコード aPair: ペア名	レバレッジデータを含む通貨レートと制限
GetCurrencyRates	aMarketCode: マーケットのコード aPair: ペア名	通貨レートと制限
GetBalances	aAccountId : アカウントのID	指定されたエクスチェンジの負荷分散
GetAccountTableData	aAccountId : アカウントのID	指定された取引所でのすべてのユーザー残高に関する情報

GetAccountLeverage	aAccountId: アカウントの ID。 aPair: ペ ア名。	アカ ウン トの レバ レッ ジに 関す る情 報
GetAccountInfo	aAccountId : アカウントの ID	単一 アカ ウン ト情 報

スマートトレード

メソッド	Arguments	説明
GetSmartTradeHistory		トレード履 歴を取得し ます。
PlaceMarketOrder	aAccountId: ア カウ ントの ID aOrderSide: 買 いまたは 売り aPair: ペア名 aQuantity: 数量	マーケッ ト注文を発注 します。
PlaceLimitOrder	aAccountId: ア カウ ントのID aOrderSide: 買 いまたは 売り aPair: 通貨ペア 名 aQuantity: 数 量 aPrice: 指値価 格	指値注文を 出します。
GetSmartTrade	ald: 取引の ID	取引 ID で スマートト レードを取 得します。
CancelSmartTrade	ald: 取引の ID	取引 ID に よってス マートト レードを キャンセル します
CloseByMarketSmartTrade	ald: 取引の ID	
EditSmartTrade	ald: 取引の ID	既存のス マートト レードを編 集します

ForceStartSmartTrade	ald: 取引の ID	スマートトレードを強制開始する
AddFundsSmartTrade	ald: 取引の ID	スマートトレードに資金を追加する
GetSmartTradeTrades	ald: 取引の ID	スマートトレードの取引を取得します。

DCA Bot

メソッド	Arguments	説明
CreateDCABot		新しい DCA ボットを作成する
GetDCABot	ald: ボットの ID	ID で DCA ボットを取得します。
GetDCABots		すべての DCA ボットを取得する
EnableDCABot	ald: ボットの ID	DCA Botを有効にする
DisableDCABot	ald: ボットの ID	DCA ボットを無効にする
DeleteDCABot	ald: ボットの ID	DCA ボットを削除する
CancelDCABot	ald: ボットの ID	DCA ボットをキャンセルする
GetDCABotStats		DCA ボット統計の取得
GetAvailableStrategyList		利用可能な戦略リストを取得します。
GetBlacklistPairs		ブラックリストペアの取得
AddBlacklistPairs		ブラックリストペアを追加

Deals

メソッド	Arguments	説明
GetDeals		すべての取引を取得する

GetDeal	aid: デीलの ID	Id で取引を取得
UpdateDeal	aid: デीलの ID	取引を更新する
CancelDeal	aid: デीलの ID	取引をキャンセルします。
CloseAtMarketDeal	aid: デीलの ID	成行価格でデールをクローズします

Grid Bot

メソッド	Arguments	説明
CreateGridBot		新しいグリッドボットを作成する
GetGridBot	aid: ボットの ID	ID でグリッド ボットを取得する
GetGridBots		すべてのGrid Botを取得します
EnableGridBot	aid: ボットの ID	グリッドボットを有効にする
DisableGridBot	aid: ボットの ID	グリッドボットを無効化する
DeleteGridBot	aid: ボットの ID	グリッドボットの削除

イベント

OnConnect

新しいWebSocket接続が開かれたとき

OnDisconnect

WebSocket接続が閉じられたとき

OnThreeCommasConnect

クライアントが 3Commas サーバーから Welcome メッセージを受信すると、接続の準備が整ったことを意味します。

OnThreeCommasConfirmSubscription

クライアントが送信した以前のサブスクリプションを確認します。

OnThreeCommasRejectSubscription

3Commasチャンネルへのサブスクライブを試みる際にエラーがあります

OnThreeCommasMessage

ここで、クライアントはサブスクライブしているチャンネルに関連して サーバーから送信されたデータを受信します

OnThreeCommasPing

サーバーからクライアントへのping。

OnThreeCommasHTTPException

HTTP REST メソッドの呼び出し中にエラーが発生した場合、このイベントが呼び出されます。

API OKX

OKX

サポートされている API

- **WebSockets API:** WebSocket サーバーに接続してリアルタイムの市場データ更新、アカウント変更、取引注文の実行を提供します。

プロパティ

WebSocket チャンネルはパブリックとプライベートの 2 つのカテゴリに分けられます。

- **パブリックチャンネル:** ティッカーチャンネル、K ラインチャンネル、指値チャンネル、注文書チャンネル、マーク価格チャンネルなどが含まれます。ログインは不要です。
- **プライベートチャンネル:** アカウントチャンネル、注文チャンネル、ポジションチャンネルなど、ログインが必要です。

OKX プロパティで以下のプロパティを設定できます。

- **ApiKey:** OKXアカウントで新しいapiキーをリクエストできます。値をこのプロパティにコピーするだけです。
- **ApiSecret:** API のシークレット値です。
- **Passphrase:** 新しい API キーを作成するときに定義するカスタム文字列です。
- **IsDemo:** 有効にすると、OKX Demoアカウントに接続します（デフォルトで無効）。
- **IsPrivate:** 有効にすると、プライベートチャンネルに接続できるようになります（デフォルトでは無効）。
- **IsBusiness:** 有効にすると、ビジネス WebSocket エンドポイント（wss://ws.okx.com:8443/ws/v5/business）に接続します。ローソク足チャンネル、アルゴ注文チャンネル、その他の高度なチャンネルに使用してください（デフォルトでは無効）。

接続

クライアントが OKX サーバーへの接続に成功すると、**OnOKXConnect** イベントが発火します。接続中にエラーが発生すると、**OnOKXError** イベントがエラーの詳細とともに発火します。

OnOKXConnect イベントが発生した後、OKXサーバーからのメッセージの送信および受信を開始できます。

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_OKX *oOKX = new TsgcWSAPI_OKX(NULL);
oOKX->Client = oClient;
oOKX->OKX->ApiKey = "alsdk23kandfnasbdfdkjhdsdf";
oOKX->OKX->ApiSecret = "aldskjfk3jkadknfajndsxfj23j";
oOKX->OKX->Passphrase = "secret_passphrase";
oClient->Active = true;
void OnOKXConnect(TObject *Sender, string aMessage, string aCode, string aRawMessage)
{
    DoLog("#OKX Connected");
}
void OnOKXError(TObject *Sender, string aCode, string aMessage, string aRawMessage)
{
    DoLog("#error: " + aMessage);
}
```

パブリックチャンネル

WebSocket フィードはオーダーと取引のリアルタイムマーケットデータ更新を提供します。WebSocket フィードにはティッカー、取引などのパブリックチャンネルがあります。

次のPublic channelsにサブスクライブできます:

メソッド	説明
SubscribeInstruments	全インストゥルメントリストはサブスクリプション後に初回プッシュされます。その後、インストゥルメントの状態に変化（FUTURES の配信、OPTION の行使、新規コントラクト/取引ペアの上場、取引停止など）があった場合にプッシュされます。
SubscribeTicker	商品の直近取引価格、bid価格、ask価格、24時間取引高を取得します。データは100msごとにプッシュされます。
SubscribeOpenInterest	オープンインタレストを取得します。データは 3 秒ごとにプッシュされます。
SubscribeCandlestick	銘柄のローソク足データを取得します。プッシュ頻度は最速インターバルの 500ms でデータをプッシュします。
SubscribeTrades	直近の約定データを取得します。約定が発生するたびにデータがプッシュされます。
SubscribeEstimatedPrices	FUTURES契約およびOPTIONの推定受渡し/権利行使価格を取得します。 推定配信/行使価格は、配信/行使の 1 時間前にプッシュされ、価格変動があった場合にプッシュされます。
SubscribeMarkPrice	マーク価格を取得します。データは、マーク価格が変化すると 200ミリ秒ごとにプッシュされ、マーク価格が変化しない場合は 10秒ごとにプッシュされます。
SubscribeMarkPriceCandlestick	マーク価格のローソク足データを取得します。データは 500 ms ごとにプッシュされます。
SubscribePriceLimit	銘柄の最大買い価格と最小売り価格を取得します。制限に変更がある場合は 5 秒ごとにデータがプッシュされ、制限に変更がない場合はプッシュされません。
SubscribeOrderBook	注文板データを取得します。 400 深さレベルには books を、5 深さレベルには book5 を、ティックバイティック 1 深さレベルには bbo-tbt を、ティックバイティック 50 深さレベルには books50-l2-tbt を、ティックバイティック 400 深さレベルには books-l2-tbt を使用します。 • books: 最初の完全スナップショットで 400 深度レベルがプッシュされます。オーダーブックに変更がある場合、増分データは 100 ms ごとにプッシュされます。 • books5: 毎回 5 深度レベルがプッシュされます。オーダーブックに変更がある場合、データは 100 ms ごとにプッシュされます。 • bbo-tbt: 毎回 1 深度レベルがプッシュされます。オーダーブックに変更がある場合、データは 10 ms ごとにプッシュされます。 • books-l2-tbt: 最初の完全スナップショットで 400 深度レベルがプッシュされます。オーダーブックに変更がある場合、増分データは 10 ms ごとにプッシュされます。 • books50-l2-tbt: 最初の完全スナップショットで 50 深度レベルがプッシュされます。オーダーブックに変更がある場合、増分データは 10 ms ごとにプッシュされます。asks または bids が空の配列の場合、50 深度ではなく 400 深度に変更があることを意味します。オーダーブックデータをローカルで維持している場合は、空の asks および bids を無視してください。
SubscribeOptionSummary	すべての OPTION 契約の詳細な価格情報を取得します。データは一度にプッシュされます。
SubscribeFundingRate	ファンディングレートを取得します。データは30秒から90秒でプッシュされます。
SubscribeIndexCandlestick	インデックスのローソク足データを取得します。データは 500 ms ごとにプッシュされます。

SubscribeIndexTicker	インデックス Ticker データを取得します
SubscribeStatus	システムメンテナンスのステータスを取得し、システムメンテナンスのステータスが変化したときにプッシュします。最初のサブスクリプション時:「最新の変更データをプッシュ」。状態が変化するたびに変更されたコンテンツをプッシュします。
SubscribePublicStructureBlockTrades	ブロックトレードが発生したときにデータがプッシュされます。
SubscribeBlockTickers	過去 24 時間の最新のブロック取引量を取得します。データはトランザクション実行イベントによってトリガーされたときにプッシュされます。また、サブスクリプションの粒度に応じて 5 分間隔でもプッシュされます。
SubscribeAllTrades	すべての取引データを取得します。取引が発生するたびにデータがプッシュされます。
SubscribeLiquidationOrders	強制清算注文を取得します。強制清算注文が発生するとデータがプッシュされます。
SubscribeADLWarning	ADL警告データを取得します。自動デレバレッジリスクが増加するとデータがプッシュされます。
SubscribeEconomicCalendar	経済カレンダーイベントを取得します。経済イベントの更新があるとデータがプッシュされます。
SubscribePublicBlockTrades	パブリックなブロックトレードを取得します。ブロックトレードがあるたびにデータがプッシュされます。
SubscribeOptionTrades	オプション取引データを取得します。オプション取引が発生するたびにデータがプッシュされます。
SubscribeCallAuctionDetails	コールオークションの詳細を取得します。コールオークション情報の更新があると、データがプッシュされます。

```
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->SubscribeInstruments(okxitFutures);
}
```

プライベートチャンネル

アカウントチャンネル、注文チャンネル、ポジションチャンネルなど、ログインが必要です。

以下のプライベートチャンネルにサブスクライブできます：

メソッド	説明
SubscribeAccount	アカウント情報を取得します。注文の発注、キャンセル、取引執行などのイベントによってトリガーされるとデータがプッシュされます。また、サブスクリプションの粒度に応じた定期的な間隔でもプッシュされます。
SubscribePositions	ポジション情報を取得します。初期スナップショットはサブスクリプションの粒度に応じてプッシュされます。注文の発注/キャンセルなどのイベントによってトリガーされたときにデータがプッシュされ、サブスクリプションの粒度に応じて定期的にもプッシュされます。

SubscribeBalanceAndPosition	アカウント残高とポジション情報を取得します。約定や資金移動などのイベントによってトリガーされたときにデータがプッシュされます。
SubscribeOrders	注文情報を取得します。最初のサブスクライブ時にはデータはプッシュされません。注文の発注/キャンセルなどのイベントによってトリガーされた場合にのみデータがプッシュされます。
SubscribeOrdersAlgo	アルゴオーダーを取得します（トリガーオーダー、OCOオーダー、条件付きオーダーを含む）。最初にサブスクライブしたときにデータはプッシュされません。注文の発注/キャンセルなどのイベントによってトリガーされた場合にのみデータがプッシュされます。
SubscribeAdvanceAlgo	高度なアルゴ注文（冰山注文、TWAP 注文、トレーリング注文を含む）を取得します。最初のサブスクライブ時にデータがプッシュされます。注文の発注/キャンセルなどのイベントによってトリガーされたときにデータがプッシュされます。
SubscribePositionRisk	このプッシュチャンネルはリスクの警告としてのみ使用され、戦略的取引のリスク判断として推奨されません。市場が激しく動いていない場合、このメッセージがプッシュされると同時にポジションが清算される可能性があります。
SubscribeAccountGreeks	アカウントのギリシャ文字情報を取得します。ポジションの増減やアカウントの現金残高などのイベントによってトリガーされたときにデータがプッシュされ、サブスクリプションの粒度に応じて定期的にもプッシュされます。
SubscribeRfqs	Rfqsを取得します。
SubscribeQuotes	クオートを取得します。
SubscribePrivateStructureBlockTrades	構造化ブロックトレードを取得します。
SubscribeSpotGridAlgoOrders	スポットグリッドのアルゴリズム注文を取得します。最初にサブスクライブしたときにデータがプッシュされます。注文の発注/キャンセルなどのイベントによってトリガーされたときにデータがプッシュされます。
SubscribeContactGridAlgoOrders	コントラクトグリッドアルゴ注文を取得します。最初にサブスクライブしたときにデータがプッシュされます。注文の発注/キャンセルなどのイベントによってトリガーされたときにデータがプッシュされます。
SubscribeGridPositions	グリッドポジションを取得します。最初にサブスクライブしたときにデータがプッシュされます。注文の発注/キャンセルなどのイベントによってトリガーされた場合にもデータがプッシュされます。
SubscribeGridSubOrders	グリッドサブオーダーを取得します。最初にサブスクライブしたときにデータがプッシュされます。注文の発注などのイベントによってトリガーされた場合にもデータがプッシュされます。
SubscribeFills	約定済み注文データを取得します。注文が約定するとデータがプッシュされます。
SubscribeDepositInfo	入金情報を取得します。入金ステータスが更新されるとデータがプッシュされます。
SubscribeWithdrawalInfo	出金情報を取得します。出金ステータスが更新されるとデータがプッシュされます。

```

TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_OKX *oOKX = new TsgcWSAPI_OKX(NULL);
oOKX->Client = oClient;
oOKX->OKX->ApiKey = "a1sdjk23kandfnasbdfdkjhsdf";
oOKX->OKX->ApiSecret = "a1dskjfk3jkadknfajndsfj23j";

```

```
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 取引には認証が必要です。

注文を行うには、十分な資金が必要です。以下にリクエストパラメータの表を示します:

パラメータ	タイプ	必須	説明
id	String	Yes	メッセージの一意識別子 クライアントが提供します。対応するリクエストを識別するためにレスポンスメッセージで返されます。 最大 32 文字の大文字小文字を区別する英数字、すべての数字、またはすべての文字の組み合わせ。
> instId	String	Yes	インストルメントID。例 <code>BTC-USD-190927-5000-C</code>
> tdMode	String	Yes	取引モード マージンモード <code>isolated</code> <code>cross</code> 非マージンモード <code>cash</code>
> ccy	String	No	証拠金通貨 <code>Single-currency margin</code> での <code>cross</code> <code>MARGIN</code> 注文にのみ適用されます。
> clOrdId	String	No	クライアント指定の注文 ID 大文字・小文字を区別する英数字の組み合わせ、数字のみ、または文字のみで最大 32 文字。
> タグ	String	No	注文タグ 大文字と小文字を区別した英数字の組み合わせ、すべて数字、またはすべて文字で最大 16 文字。
> サイド	String	Yes	注文側、 <code>buy</code> <code>sell</code>
> posSide	String	オプション	ポジションサイド デフォルトは <code>net</code> モードでは <code>net</code> です <code>long/short</code> モードでは必須であり、 <code>long</code> または <code>short</code> のみが指定できます。 <code>FUTURES/SWAP</code> にのみ適用されます。
> ordType	String	Yes	注文タイプ <code>market</code> : 成行注文 <code>limit</code> : 指値注文 <code>post_only</code> : Post-only注文 <code>fok</code> : Fill-or-kill注文 <code>ioc</code> : Immediate-or-cancel注文 <code>optimal_limit_ioc</code> : Immediate-or-cancelを伴う成行注文
> sz	String	Yes	購入または販売する数量。
> px	String	オプション	価格 <code>limit</code> , <code>post_only</code> , <code>fok</code> , <code>ioc</code> 注文にのみ適用されます。
> reduceOnly	Boolean	No	ポジションのみを縮小するかどうか。 <code>true</code> <code>false</code> 。デフォルトは <code>false</code> 。

パラメータ	タイプ	必須	説明
			MARGIN 注文、および net モードの FUTURES/SWAP 注文にのみ適用 Single-currency margin および Multi-currency margin にのみ適用
> tgtCcy	String	No	数量タイプ base_ccy : 基軸通貨、quote_ccy : 決済通貨 Market注文で取引される SPOT にのみ適用されます デフォルトは買いの場合 quote_ccy 、売りの場合 base_ccy
> banAmend	Boolean	No	スポット注文の修正をBANするかどうか、trueまたはfalse、デフォルトはfalseです。 banAmendがtrueの場合、残高が不足していると注文の発注に失敗します。 SPOTのmarket orderにのみ適用されます

注文発注の例

注文は十分な資金がある場合にのみ行えます。

```
// Place Martket Order
TsgcWSAPI_OKX1->PlaceMarketOrder(okxosBuy, "ETH-BTC", 1);
// Place Limit Order
TsgcWSAPI_OKX1->PlaceLimitOrder(okxosBuy, "ETH-BTC", 1, 0.25);
```

注文キャンセルの使用例

未完了の注文をキャンセルします。

```
TsgcWSAPI_OKX1->CancelOrder("
ETH-BTC
", "457589362405027840");
```

注文の修正

未完了の注文を変更します。

```
TsgcWSAPI_OKX1->AmendOrder("ETH-BTC", "457589362405027840", "", 2);
```

Batch Trade Operations

WebSocket Trade API は、複数の注文を一度に発注、キャンセル、修正するためのバッチ操作もサポートしています。これらの操作には認証が必要です。

メソッド	説明
BatchPlaceOrders	1回のリクエストで複数の注文を発注します。一度に最大 20 件の注文を発注できます。
BatchCancelOrders	1回のリクエストで複数の注文をキャンセルします。一度にキャンセルできる注文の最大数は20件です。
BatchAmendOrders	1回のリクエストで複数の未完了注文を修正します。一度に最大20件まで修正できます。
MassCancelOrders	特定のインストゥルメントタイプのすべての保留中の注文を一括キャンセルします。

API XTB

XTB

サポートされている API

- **WebSockets API:** WebSocket サーバーに接続し、リアルタイムの市場データ更新、アカウント変更、取引注文の発注を提供します。

プロパティ

WebSocket プロトコルは2種類のリクエストを許可します: **Streaming コマンド**(ライブ更新を受信)および **Retrieve Trading Data**(何らかの情報を取得するリクエストをサーバーに送信)。

XTB プロパティで以下のプロパティを設定できます。

- **User:** 接続を識別するユーザー名。
- **Password:** ユーザーのシークレット値。
- **Demo:** 有効にすると XTB デモアカウントに接続します（デフォルトでは無効）。

接続

クライアントがXTBサーバーに正常に接続すると、**OnXTBConnect** イベントが発生します。接続を試みる際にエラーが発生した場合は、エラーの詳細とともに**OnXTBError** イベントが発生します。

OnXTBConnect イベントが発生した後、XTBサーバーからのメッセージの**送信**および**受信**を開始できます。

```
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);
}
```

接続コマンド

メソッド	説明
ログイン	クライアントアプリケーションは何らかのアクションを実行するためにログインプロセスを実行する必要があります。適切なログインプロセスの前には機能が利用できません。ログインメソッドはクライアントが WebSocket サーバーに接続し、User/Password の値が設定された後に自動的に呼び出されます。
Logout	

ストリーミングコマンド

以下のチャンネルをサブスクライブできます：

メソッド	説明
SubscribeBalance	システムで利用可能になり次第、実際のアカウント指標の値をリアルタイムで取得できます。
SubscribeCandles	API チャートキャンドルのサブスクライブとアンサブスクライブを行います。各キャンドルの間隔は 1 分です。新しいキャンドルは 1 分ごとに届きます。
SubscribeKeepAlive	「キープアライブ」メッセージを購読および購読解除します。新しい「キープアライブ」メッセージは API によって 3 秒ごとに送信されます。
SubscribeNews	ニュースのサブスクライブおよびサブスクライブ解除を行います。
SubscribeProfits	利益の購読および購読解除を行います。
SubscribeTickPrices	相場のサブスクリプションを確立し、システムで利用可能になり次第リアルタイムで関連情報を取得できます。getTickPrices コマンドは同じシンボルに対して何度も呼び出せますが、特定のシンボルのサブスクリプションは 1 つだけ作成されます。複数のレコードが利用可能な場合、受信順序は保証されません。
SubscribeTrades	ユーザーの取引ステータスデータのサブスクリプションを確立し、システムで利用可能になり次第リアルタイムで関連情報を取得できます。複数のレコードが利用可能な場合、受信順序は保証されませんのでご注意ください。
SubscribeTradeStatus	送信されたトレードリクエストのステータスをリアルタイムで、システムで利用可能になり次第取得できます。複数のレコードが利用可能な場合、受信順序は保証されないことにご注意ください。
SubscribePing	この関数を定期的に呼び出すだけで、システム内のすべてのコンポーネントの内部状態を更新するのに十分です。ストリーミング接続では、クライアントがセッション内でコマンドを送信しない場合、一方向のネットワークトラフィックのみが生成されます。他のコマンドを実行しないアプリケーションは、少なくとも 10 分に 1 回はこのコマンドを呼び出すことをお勧めします。

取引データの取得

次のリクエストを送信できます：

メソッド	説明
GetAllSymbols	ユーザーが利用可能なすべてのシンボルの配列を返します。
GetCalendar	市場イベントのカレンダーを返します。
GetChartLastRequest	この機能は通常、現在のローソク足データを取得するための優先される方法であるストリーミング同等の getCandles に置き換えることができます。開始日から現在まで、チャート情報を返します。 CHART_LAST_INFO_RECORD で選択した期間が1分を超える場合、API によって返される最後のローソク足は期間の終了まで変化する可能性があります（ローソク足は毎分自動的に更新されます）。
GetChartRangeRequest	この関数は通常、現在のローソク足データを取得する際に推奨されるストリーミング等価物 getCandles で置き換えられることにご注意ください。指定された開始日と終了日の間のデータを含むチャート情報を返します。

GetCommissionDef	手数料と為替レートの計算結果を返します。値は期待値として計算されるため、完全に正確ではない可能性があります。
GetCurrentUserData	アカウントの通貨、アカウントのレバレッジに関する情報を返します。
GetIbsHistory	指定した時間範囲から IB データを返します。
GetMarginLevel	この関数は通常、ストリーミング版の getBalance で置き換えられることに注意してください。getBalance はアカウント指標を取得する推奨方法です。さまざまなアカウント指標を返します
GetMarginTrade	指定されたインストゥルメントとボリュームの予想マージンを返します。値は予想マージン値として計算されるため、完全に正確ではない場合があります。
GetNews	この関数は通常、ニュースデータ取得の推奨方法であるストリーミング版の getNews に置き換えられることに注意してください。指定された期間内に取引サーバーから送信されたニュースを返します。
GetProfitCalculation	指定した取引データに基づいて推定利益を計算します。計算機のようなアプリにのみ使用してください。オープン中の取引の利益は、サーバーの計算精度が高いためサーバーから取得してください。
GetServerTime	取引サーバーの現在時刻を返します
GetStepRules	DMA のステップルールリストを返します。
GetSymbol	ユーザーが利用可能なシンボルに関する情報を返します。
GetTickPrices	この関数は通常、ストリーミングの同等版である getTickPrices で置き換えることができ、これがティックデータを取得する推奨される方法であることに注意してください。指定されたシンボルの現在のクォートの配列を返します。指定されたタイムスタンプから変更されたクォートのみが返されます。出力から得られた新しいタイムスタンプは、このコマンドの次の呼び出しの引数として使用されます。
GetTradeRecords	orders 引数に一覧されている注文の取引配列を返します
GetTrades	この関数は通常、取引データを取得する優先方法であるストリーミング対応の getTrades に置き換えられることに注意してください。ユーザーの取引の配列を返します。
GetTradesHistory	この関数は通常、ストリーミング版の getTrades で置き換えることができます。これが取引データを取得する推奨の方法です。指定した期間内にクローズされたユーザーの取引の配列を返します。
GetTradingHours	クォートと取引時間を返します。
GetVersion	現在の API バージョンを返します。
Ping	この関数を定期的呼び出すだけで、システム内のすべてのコンポーネントの内部状態を更新するのに十分です。他のコマンドを実行しないアプリケーションは、少なくとも 10 分に 1 回はこのコマンドを呼び出すことが推奨されます。この関数のストリーミング相当物は ping と getKeepAlive の組み合わせであることに注意してください。
TradeTransaction	取引トランザクションを開始します。tradeTransaction はメイントランザクション情報をサーバーに送信します。
TradeTransactionStatus	この関数は通常、トランザクションステータスデータを取得する推奨方法であるストリーミング相当の getTradeStatus で置き換えることができることに注意してください。現在のトランザクションステータスを返します。トランザクション処理のどの時点でも、クライアントはサーバー側でトランザクションのステータスを確認できます。そのためには、クライアントは tradeTransaction 呼び出しから取得した一意の order を提供する必要があります。

API Bybit

Bybit

サポートされている API

- **WebSocket API:** WebSocket サーバーに接続し、リアルタイムの市場データ更新、アカウント変更などを提供します。
- **REST API:** HTTPリクエストを送信して、市場データの取得、注文の発注、アカウントデータの取得などを行います。

現在サポートされている API バージョンは V5 です。V5 API は Bybit の製品ライン全体に統一性と効率性をもたらし、スポット、デリバティブ、オプションを 1 つの仕様セットに統合しています。

OpenAPI バージョン	アカウント タイプ	リニア		逆数			ス ポ ッ ト	オ プ シ ョ ン
		USDT Perpetual	USDC 無期限	USDC Futures	無期限 (Perpetual)	先 物		
V5	統合取引 アカウント	✓	✓	✓	注意を参照		✓	✓
	クラ シック アカ ウント	✓			✓	✓	✓	
V3	統合取引 アカウント	✓	✓					✓
	クラ シック アカ ウント	✓			✓	✓	✓	

*注意: ユニファイドアカウントはインバース取引をサポートしています。ただし、使用されるマージンはユニファイドウォレットではなく、インバースデリバティブウォレットからです。

プロパティ

Bybit プロパティで次のプロパティを設定できます。

- **ApiKey:** Bybit アカウントで新しい API キーをリクエストできます。値をこのプロパティにコピーしてください。APIKey が設定されている場合、クライアントは WebSocket プライベートサーバーに接続します。空の場合は、WebSocket パブリックサーバーに接続します。
- **ApiSecret:** API のシークレット値です。
- **SignatureExpires:** 署名が期限切れになるまでの秒数（デフォルトは 10 秒）。
- **TestNet:** 有効にすると、Bybit TestNet デモアカウントに接続します（デフォルトでは無効）。

接続

クライアントがBybitサーバーに正常に接続すると、**OnConnect** イベントが発生します。**OnConnect** イベントが発生した後、Bybitサーバーとのメッセージの**送信**および**受信**を開始できます。プライベートWebSocketチャンネルに接続する場合は、いずれかのチャンネルにサブスクライブする前に、**OnBybitAuthentication** イベントが発生するまで待機し、successパラメータがtrueかどうかを確認する必要があります。

クライアントは複数のAPIをサポートしているため、BybitClientプロパティを使用して使用するAPIを設定してください。

- **bybSpot**
- **bybInverse**
- **bybLinear**
- **bybPerpetual**

WebSocket スポットプライベート API への接続例を以下に示します。

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bybit *oBybit = new TsgcWSAPI_Bybit(NULL);
oBybit->Client = oClient;
oBybit->Bybit->ApiKey = "a1sdjk23kandfnasbdfdkjhdsf";
oBybit->Bybit->ApiSecret = "a1dskjfk3jkadknfajndsjsfj23j";
oBybit->BybitClient = bybSpot;
oClient->Active = true;
void OnConnect(TsgcWSConnection *Connection)
{
    DoLog("#Bybit Connected");
}
```

Spot WebSocket Serverへの接続が成功したら、WebSocketチャンネルのサブスクライブを開始できます。**SPOT** プロパティにアクセスして、利用可能なsubscribe/unsubscribeメソッドのいずれかを呼び出すだけです。

イベント

bybit クライアントは、接続フローを制御し WebSocket サーバーから送信されたデータを取得するための以下のイベントを実装しています：

- **OnConnect:** WebSocket クライアントが WebSocket サーバーに接続したときに発生します。
- **OnDisconnect:** WebSocket クライアントが WebSocket サーバーから切断されたときに発生します。
- **OnBybitAuthentication:** クライアントがPrivate WebSocket Serverに対して認証を行ったときに発生します。
- **OnBybitSubscribe:** クライアントがWebSocketチャンネルにサブスクライブしたとき。
- **OnBybitUnSubscribe:** クライアントが WebSocket チャンネルからサブスクライブ解除したとき。
- **OnBybitData:** クライアントがサーバーからデータを受信したとき。
- **OnBybitError :** bybit WebSocket 接続中にエラーが発生したとき。
- **OnBybitHTTPException:** REST リクエスト中にエラーが発生した場合に発生します。

WebSocket API

WebSocket フィードはオーダーと取引のリアルタイムマーケットデータ更新を提供します。WebSocket フィードにはティッカー、取引などのパブリックチャンネルがあります。

以下のチャンネルにサブスクライブできます：

メソッド	パブリックまたはプライベート	説明

SubscribeOrderBook	パブリック	オーダーブックストリームをサブスクライブします。さまざまな深度をサポートします。
SubscribeTrade	パブリック	最新の取引ストリームにサブスクライブします。
SubscribeTicker	パブリック	ティックーストリームをサブスクライブします。
SubscribeKLine	パブリック	klines ストリームを購読します。
SubscribeLiquidation	パブリック	清算ストリームをサブスクライブします
SubscribeLT_KLine	パブリック	レバレッジドトークンの K ライン（ローソク足）ストリームをサブスクライブします。
SubscribeLT_Ticker	パブリック	レバレッジドトークンのティックーストリームをサブスクライブします。
SubscribeLT_Nav	パブリック	レバレッジドトークンのティックーストリームをサブスクライブします。
SubscribePosition	プライベート	レバレッジドトークンの純資産価値（NAV）ストリームをサブスクライブします。
SubscribeExecution	プライベート	サブスクライブ
SubscribeOrder	プライベート	サブスクライブ
SubscribeWallet	プライベート	サブスクライブ
SubscribeGreek	プライベート	サブスクライブ
SubscribeDcp	プライベート	サブスクライブ
SubscribeInsurance	パブリック	保険基金ストリームをサブスクライブします。
SubscribeOrderPriceLimit	パブリック	注文価格制限ストリームをサブスクライブします。
SubscribeADLAlert	パブリック	自動デレバレッジアラートストリームをサブスクライブします。
SubscribeFastExecution	プライベート	高速実行ストリームをサブスクライブします。

認証成功後にプライベート WebSocket チャンネルをサブスクライブする例を以下に示します。

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bybit *oBybit = new TsgcWSAPI_Bybit(NULL);
oBybit->Client = oClient;
oBybit->Bybit->ApiKey = "alsdkj23kandfnasbdfdkjhdsdf";
oBybit->Bybit->ApiSecret = "alsdkjfk3jkadknfajndsxfj23j";
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 APIには、市場、プライベートアカウント、ウォレットからデータを要求するためのパブリックメソッドとプライベートメソッドのリストがあります。利用可能なメソッドのリストを以下に示します。

メソッド	パブリック / プライベート
GetServerTime	パブリック
GetKLine	パブリック
GetMarkPriceKLine	パブリック
GetIndexPriceKLine	パブリック
GetPremiumIndexPriceKLine	パブリック
GetInstrumentsInfo	パブリック
GetOrderBook	パブリック
GetTickers	パブリック
GetFundingRateHistory	パブリック
GetPublicRecentTradingHistory	パブリック
GetOpenInterest	パブリック
GetHistoricalVolatility	パブリック
GetInsurance	パブリック
GetRiskLimit	パブリック
GetDeliveryPrice	パブリック
GetLongShortRatio	パブリック
PlaceOrder	プライベート
PlaceMarketOrder	プライベート
PlaceLimitOrder	プライベート
AmendOrder	プライベート
CancelOrder	プライベート
GetOpenOrders	プライベート
CancelAllOrders	プライベート
GetOrderHistory	プライベート
GetPositionInfo	プライベート
SetLeverage	プライベート
SwitchCrossIsolatedMargin	プライベート
SetTPSLMode	プライベート
SwitchPositionMode	プライベート
SetRiskLimit	プライベート
SetTradingStop	プライベート
SetAutoAddMargin	プライベート
AddOrReduceMargin	プライベート
GetExecution	プライベート
GetClosedPNL	プライベート
ConfirmNewRiskLimit	プライベート
GetWalletBalance	プライベート
GetAccountInfo	プライベート
GetTransactionLog	プライベート
BatchPlaceOrder	プライベート
BatchAmendOrder	プライベート
BatchCancelOrder	プライベート
SetDCP	プライベート
GetFeeRate	プライベート

GetCollateralInfo	プライベート
SetMarginMode	プライベート
GetBorrowHistory	プライベート
GetCoinGreeks	プライベート
GetCoinInfo	プライベート
GetAllCoinsBalance	プライベート
CreateInternalTransfer	プライベート
GetInternalTransferList	プライベート
GetDepositRecords	プライベート
GetDepositAddress	プライベート
CreateWithdrawal	プライベート
CancelWithdrawal	プライベート
GetWithdrawalRecords	プライベート

オープン注文を取得する例を以下に示します。

```
TsgcWebSocketClient *oClient = new TsgcWebSocketClient();
TsgcWSAPI_Bybit *oBybit = new TsgcWSAPI_Bybit(NULL);
oBybit->Client = oClient;
oBybit->Bybit->ApiKey = "a1sdjk23kandfnasbdfdkjhdsf";
oBybit->Bybit->ApiSecret = "aldskjfk3jkadknfajndsjsfj23j";
oBybit->BybitClient = bybSpot;
oBybit->REST_API->GetAccountInfo();
```

API Forex

Forex.com (StoneX / CityIndex)

Forex.com は StoneX が運営するリテール取引所です。このコンポーネントは Forex.com、StoneX、CityIndex (すべて同じプラットフォームを共有) が使用する取引 REST API と Lightstreamer ベースのストリーミングサービスをラップします。以下の API がサポートされています:

1. **Streaming API:** WebSocket上のLightstreamer TLCP 2.5。価格ティック、注文更新、ポジション更新、口座証拠金、相場は、REST LogOnの成功後にリアルタイムでプッシュされます。
2. **REST API:** セッションログイン(UserName + Password + AppKey)、取引およびstop-limit注文管理、口座クエリ、市場データ。HTTPSを使用します。

コンポーネント

- **TsgcWSAPI_Forex:** メインクライアント。Credentialsを保持し、REST_API、内部のLightstreamerストリーミングクライアント、およびOnForex*イベントを公開します。
- **TsgcHTTP_API_Forex:** REST コンポーネント。メインクライアントの REST_API プロパティを通じてアクセスできます。
- **TsgcWSPClient_Lightstreamer:** PRICES / ORDERS / QUOTES / CLIENTACCOUNTMARGIN / TRADEMARGIN のストリーミングサブスクリプションに使用される内部 TLCP 2.5 サブプロトコルクライアント。

公開プロパティ

- **Credentials:** REST APIにログインするために必要なフィールドのグループ。
 - **UserName:** Forex.com / StoneX / CityIndexアカウントのユーザー名。
 - **Password:** アカウントのパスワード。
 - **AppKey:** 取引所が発行するアプリケーションキー。
 - **AppVersion:** オプションのクライアントバージョン文字列。
 - **AppComments:** ログイン時に記録される自由形式のテキスト。
 - **StreamingHost:** Lightstreamerホスト (デフォルトは公開ストリーミングホスト)。
- **AutoReconnect:** 有効にすると、下層のWebSocketクライアントWatchDogへ転送し、ネットワーク切断後にストリーミングセッションが自動的に再接続されるようにします (デフォルトはtrue)。
- **REST_API:** TsgcHTTP_API_Forex コンポーネントへの参照。
- **Lightstreamer:** 内部の TsgcWSPClient_Lightstreamer サブプロトコルクライアントへの参照 (AdapterSet は STREAMINGALL に固定)。
- **SessionToken:** 読み取り専用。REST LogOn 呼び出しによって返されるセッショントークン。Lightstreamer の LS_password として使用されます。
- **ClientAccountId / TradingAccountId:** 読み取り専用。ログオン成功後に設定されます。

公開イベント

OnForexConnect

ストリーミングセッションが作成され、サブスクリプションを開始できるようになったときに発火します。

OnForexDisconnect

ストリーミングセッションが終了したとき (サーバーのクローズ、ネットワーク切断、または明示的な Disconnect) に発火します。

OnForexException

REST エラー (LogOn の失敗を含む) およびストリーミングセッションの障害時に発生します。

OnForexPriceTick

すべての PRICES 更新に対して発生します: Bid、Offer、Price、Change、Direction、AuditId、TickDate。

OnForexOrderUpdate

すべての ORDERS 更新 (新規、約定、キャンセル、拒否) に対して発生します。

OnForexPositionUpdate

アクティブな TradingAccount のすべての未決ポジション更新に対して発火します。

OnForexAccountMargin

CLIENTACCOUNTMARGIN / TRADEMARGIN の更新 (エクイティ、マージン、フリーマージン、通貨) ごとに発生します。

OnForexQuote

すべての QUOTES 更新（見積もりリクエストワークフロー）に対して発火します。

OnForexSubscribe

サーバーがLightstreamerサブスクリプション（SUBOK / SUBCMD）を確認するたびに発生します。
パラメータ: *aAdapter*（PRICES、ORDERS、QUOTES、CLIENTACCOUNTMARGIN または TRADEMARGIN）、*aGroup*（アイテムグループ、例: *ID.<ClientAccountId>*）、*altems*、*aFields*。

OnForexUnsubscribe

UnwatchMarket、ストリーミング切断、またはサーバーが UNSUB フレームを送信するなどにより、サブスクリプションが終了したときに発生します。パラメータ: *aAdapter*、*aGroup*。

最も一般的な使用方法

- REST API
 - [REST API への接続方法](#)
 - [取引注文およびストップリミット注文の発注方法](#)
 - [口座、ポジション、注文履歴の照会方法](#)
 - [マーケットデータの取得方法](#)
- WebSockets API
 - [WebSocket API への接続方法](#)
 - [ストリーミングアダプターへのサブスクライブ方法](#)

サンプル

クライアントを作成し、認証情報を設定し、接続して、市場価格ストリームをサブスクライブして切断します。

```
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 | REST API に接続する

Forex.com / StoneX / CityIndex Trading API へのすべての呼び出しにはアクティブなセッションが必要です。セッション状態は **LogOn** で取得し、**LogOff** で終了し、**Ping** で維持します。

資格情報

メインクライアントコンポーネントに認証情報を設定します。最初の呼び出し時にRESTセッションにコピーされます。

- **UserName:** Forex.com / StoneX / CityIndexアカウントのユーザー名。
- **Password:** アカウントパスワード。
- **AppKey:** 取引所から発行されたアプリケーションキー。
- **AppVersion / AppComments:** ログイン時に記録されるオプションの文字列。

セッションメソッド

メソッド	パラメータ	説明
LogOn	UserName, Password, AppKey	REST セッションを開きます。完全な JSON レスポンスを返し、メインクライアントに SessionToken、ClientAccountId、TradingAccountId を格納します。以降のすべての REST およびストリーミング呼び出しはそのトークンで認証されます。
LogOff		サーバーのセッションを無効にしてトークンをクリアします。
Ping		セッションを維持します。サーバーセッションは一定期間操作がないと期限切れになります。他の REST 呼び出しが行われていない場合は、定期的に Ping を発行してください。
GetServiceStatus		現在のサービスステータスを報告します（認証は不要です）。

Token Flow

LogOn が返すトークンは、その後のすべての REST リクエストに *UserName* HTTP ヘッダーとして付与されます。同じトークンは、Lightstreamer セッションが作成される際に *LS_password* フィールドとしてストリーミングクライアントによっても使用されます。LogOn を 2 回呼び出す必要はありません。メインクライアントで **Connect** を呼び出すと、最初に REST LogOn を実行し、その後ストリーミングソケットを開きます。

サンプル

REST セッションを開いて Ping を呼び出す

```
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 APIは5つの注文管理エンドポイントを公開しています。各エンドポイントは3つのオーバーロード形式で提供されており、呼び出し元が必要な制御レベルを選択できます。

Endpoints

メソッド	説明
NewTradeOrder	CFD マーケットにマーケット/ベストレート取引注文を発注します。
UpdateTradeOrder	既存のオープントレードにストップ/リミット保護をアタッチまたは変更します。
NewStopLimitOrder	保留中のストップまたはリミット注文（まだ市場に出ていない）を発注します。
UpdateStopLimitOrder	既存の保留中のストップまたは指値注文を修正します。
CancelOrder	保留中のストップまたは指値注文をキャンセルします。

オーバーロードティア

注文メソッドは2つのオーバーロードで利用できます。

1. **スカラーパラメーターオーバーロード**（NewTradeOrder および CancelOrder のみ）：最もよく使用されるフィールド（MarketId、TradingAccountId、Direction、Quantity、BidPrice / OfferPrice、NewTradeOrder の AuditId、CancelOrder の OrderId、TradingAccountId、MarketId）を受け取ります。ヘルパーは内部でリクエストを構築します。シンプルな保護なし注文に便利です。
2. **型付き注文オブジェクトオーバーロード** — TsgcHTTPForexTradeOrder、TsgcHTTPForexStopLimitOrder、または TsgcHTTPForexCancelOrder を受け取ります。これらのオブジェクトは、IfDone（添付されたストップ/リミットブラケット）、Close 配列（注文 ID によるクローズセマンティクス）、PositionMethodId、Reference、部分約定フラグを含むすべてのドキュメント化されたフィールドを公開します。これは単純なマーケット注文を超えるものに対して推奨されるパスです。オブジェクト構築の規則は Kucoin などの他のベニューと共有されています。

TsgcHTTPForexTradeOrder フィールド

- **MarketId**: ListCfdMarkets または FullSearchWithTags によって返される数値マーケット ID。
- **TradingAccountId**: アクティブな取引アカウントです。LogOnの後にメインクライアントから読み取ってください。
- **Direction**: 買いまたは売り。
- **Quantity**: マーケットのネイティブ単位での取引サイズ。
- **BidPrice / OfferPrice**: 注文が送信されたときにクライアントが観測した価格です（Forex.com は古いクオートを検出するためにこれらを使用します）。
- **AuditId**: 最新の価格ティックからの価格監査 ID。
- **PositionMethodId**: ネットティング/ヘッジング動作（1 = ネットティング、2 = ヘッジング）。
- **IfDone**: 親注文が約定したときにアクティブになる、アタッチされた注文（ストップ、指値、保証付きストップ、トレーリングストップ）のリストです。
- **Close**: この注文が約定したときにクローズする注文IDの配列（部分クローズ）。
- **Reference**: 呼び出し元が指定するクライアント参照。注文の更新でそのままエコーバックされます。

サンプル

テイクプロフィットとストップロスを付けた市場でのEUR/USD 1000買い

```
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 | アカウントクエリ

これらのエンドポイントはアカウントの状態と注文履歴を読み取ります。アクティブなセッションが必要です ([Connect REST API](#) を参照)。

Endpoints

メソッド	パラメータ	説明
GetClientAndTradingAccount		認証済みユーザーにリンクされたクライアントアカウントと取引アカウントのリストを返します。
ListOpenPositions	TradingAccountId	取引アカウントの現在のすべてのオープンポジションを返します。
ListActiveStopLimitOrders	TradingAccountId	すべての保留中のストップ注文と指値注文を返します。
GetOrder	OrderId	IDで単一の注文を返します。約定履歴と添付のIfDone注文を含みます。
ListTradeHistory	TradingAccountId, MaxResults	アカウントの取引注文履歴を最新のものから順に返します。
ListStopLimitOrderHistory	TradingAccountId, MaxResults	アカウントのストップリミット注文履歴を最新のものから順に返します。
SimulateTrade	JSON / スカラー / order-object	注文を送信せずにサーバーバリデーターで検証します。事前のマージンおよび価格許容チェックに役立ちます。NewTradeOrderと同じ3つのオーバーロード層を受け付けます。

廃止されたエンドポイント

公式のForex.com / StoneX / CityIndexのリファレンスには、ライブサーバーがもはや提供していない一握りのエンドポイントが依然として記載されています。これらは意図的にコンポーネントによって公開されていません。

- **GetSystemLookup** — リファレンスは汎用ルックアップエンドポイントを説明していますが、サーバーはすべての既知のルックアップキーに対してHTTP 404 で応答します。代わりに **GetClientAndTradingAccount** と **ListCfdMarkets** を使用します。
- **/news/headlines/*** — ニュースフィードファミリーはリファレンスにまだ記載されていますが、本番サーバーではHTTP 500 を返します。Trading API を通じた代替手段はありません。ニュースはウェブフロントエンドのみで提供されます。

サンプル

オープンポジションと直近の50件の取引を一覧表示します

```
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 | マーケットデータ

静的なマーケットメタデータと過去の価格。これらのエンドポイントは、トレードエンドポイントおよびストリーミングアダプターに渡す **MarketId** 値を検索するために使用されます。

Endpoints

メソッド	パラメータ	説明
GetMarketInformation	MarketId	完全な市場ディスクリプター（シンボル、小数桁数、ピップサイズ、マージンファクター、取引時間）を返します。
ListCfdMarkets	SearchByMarketName, ClientAccountId, MaxResults	名前プレフィックスに一致する CFD 市場を一覧表示します。
FullSearchWithTags	Query、Tags、MaxResults	オプションのタグフィルター（FX、インデックス、株式、商品）を使用してすべての市場にわたるフリーテキスト検索。
GetPriceBars	MarketId, Interval, Span, PriceBars	過去の OHLC バー。Interval は MINUTE、HOUR、DAY、WEEK のいずれかで、Span はバーごとの間隔数です。
GetPriceTicks	MarketId, PriceTicks	市場の最新ティックスナップショット（bid / ask / mid）。

サンプル

EUR/USD を検索し、最新の 100 本の 1 分足バーを読み取ります。

```
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 | WebSocket API への接続

Forex.com/StoneX/CityIndex は Lightstreamer を通じてマーケットデータとアカウントアップデートをストリーミングします。コンポーネントは WebSocket 上でネイティブの Lightstreamer TLCP 2.5 ワイヤプロトコルを使用します。Lightstreamer のネイティブライブラリは不要です。

トランスポート

- **WebSocket サブプロトコル:** *TLCP-2.5.0.lightstreamer.com*。Sec-WebSocket-Protocol リクエストヘッダーでアドバタイズされます。
- **アダプターセット:** *STREAMINGALL*。内部の **Lightstreamer** サブプロトコルクライアントで設定されます。Forex.com のすべてのデータアダプター (PRICES、ORDERS、QUOTES、CLIENTACCOUNTMARGIN、TRADEMARGIN) はこのアダプターセット下にあります。
- **Authentication:** REST **LogOn**呼び出しから取得したセッショントークンは、Lightstreamerセッションの作成時に *LS_password*として送信され、アカウントのユーザー名に設定された *LS_user*も一緒に送信されます。

再接続

AutoReconnect が有効 (デフォルト) の場合、基盤となる WebSocket クライアントの WatchDog がネットワークエラー後にソケットを再度開きます。Lightstreamer プロトコル自体は、サーバーがクライアントを別の URL に移動するかセッションを再バインドさせたいときに *LOOP* フレームをシグナルします。コンポーネントはこれらを透過的に処理し、完全なセッション切断の前後にのみ **OnForexDisconnect/OnForexConnect** を発生させます。REST セッションが期限切れになると (しばらく Ping がない場合)、ストリーミングサーバーがセッションを閉じます。コンポーネントは保存された認証情報を使用して再度 **LogOn** を呼び出し、**SessionToken** をリフレッシュし、新しいトークンで Lightstreamer セッションを再開します。

サンプル

ストリーミングセッションを開始します

```
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 | ストリーミングアダプターをサブスクライブする

ストリーミングデータは、**STREAMINGALL**アダプターセット配下の5つのLightstreamerデータアダプターを通じて配信されます。各アダプターは独自のアイテム命名規則とサブスクリプションモードを持ちます。スナップショット形式の状態（現在価格、現在のアカウントマージン）には**MERGE**、イベントストリームチャンネル（注文更新、ポジション更新、クォートイベント）には**RAW**が使用されます。

アダプター

DataAdapter	アイテム	Mode	イベント	フィールド
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

注意:

- PRICES と CLIENTACCOUNTMARGIN は統一された **ID.{numericId}** アイテム形式を使用し、アイテムごとに単一の可変レコードをストリーミングします（MERGE）。
- ORDERS、QUOTES、TRADEMARGINはレガシーのアダプターごとのプレフィックス（**ORDERS.{id}**、**QUOTES.{id}**、**TRADEMARGIN.{id}**）を使用し、イベントのファイアホースをストリームします（RAW）。1つのアイテムがログインした**ClientAccountId**のすべての注文/見積み/取引を保持し、レコードは**OrderId/QuoteId**によって識別されます。
- TRADEMARGINレコードは、レガシーなCityIndex SDKの公式**TradeMarginDTO**に対応します。**DirectionId**は数値コードとして到着します（1 = buy、2 = sell）。ライブラリは**Direction**を文字列**"buy"/"sell"**としても公開します。**OpenPrice**は取引のエントリ価格（サーバーフィールド**PriceTakenAt**）、**CurrentPrice**は最新の評価価格（**PriceCalculatedAt**）、**OpenTradeEquity**はアカウント通貨建ての変動損益（**OTEConverted**）です。

- ORDERSの**Direction**フィールドはサーバーが公開する数値コード (1 / 2) です。**Status**は文字列 (例: "Accepted"、"Open"、"Cancelled") です。**OrderType**は文字列 ("Market" / "Stop" / "Limit") です。
- TRADEMARGIN フィールドはトレードごとの **TradingAccountId**、**MarketName**、またはトレードごとのタイムスタンプを公開しません。1つのアイテムがログインした **ClientAccountId** のすべてのポジションを持ちます。これらのフィールドには **REST_API.ListOpenPositions** を呼び出してください。ORDERS フィールドは **TriggerPrice** や **Applicability** を公開しません。必要な場合は **REST_API.GetOrder** を呼び出してください。

ヘルパーメソッド

メインクライアントは Lightstreamer の SUB / UNSUB の仕組みをラップします。主なものは以下の通りです。

- **WatchMarket(aMarketId)** : 1つのマーケットの PRICES にサブスクライブします。更新は **OnForexPriceTick** で届きます。
- **UnwatchMarket(aMarketId)**: サブスクリプションを削除します。
- **WatchAccount** : ログイン済みアカウントの ORDERS、CLIENTACCOUNTMARGIN、TRADEMARGIN をサブスクライブします。更新は対応するイベントに届きます。
- **WatchQuotes**: リクエストフォークアウトのワークフロー用に QUOTES アダプターをサブスクライブします。

サブスクリプションライフサイクルイベント

メインクライアントは、基盤となる TLCP SUBOK / UNSUB フレームを反映する 2つのライフサイクルイベントを発生させます:

- **OnForexSubscribe(Sender, aAdapter, aGroup, altems, aFields)**: サーバーが Lightstreamer サブスクリプションを確認したときに発生します。*aAdapter* はデータアダプター名 (PRICES、ORDERS、QUOTES、CLIENTACCOUNTMARGIN、TRADEMARGIN)。*aGroup* はアイテムグループ (例: *ID.<ClientAccountId>* または *ORDERS.<ClientAccountId>*)。 *altems* と *aFields* は SUBOK / SUBCMD フレームでサーバーが報告したカウントです。
- **OnForexUnsubscribe(Sender, aAdapter, aGroup)** : **UnwatchMarket** の呼び出し、ストリーミング切断時、またはサーバーが UNSUB フレームを送信したときにサブスクリプションが終了すると発生します。

サンプル

EUR/USD の価格ティックをサブスクライブする

```
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 API を使用すると、開発者は新しいトランザクションとブロックに関するリアルタイム通知を受け取ることができます。

WebSocket が開いたら、チャンネルをサブスクライブできます:

- **SubscribeTransactions:** すべての新しいビットコイントランザクションの通知をサブスクライブします。
- **UnsubscribeTransactions:** すべての新しいビットコイントランザクションの通知をサブスクライブ解除します。
- **SubscribeAddress:** 特定のビットコインアドレスの新しいトランザクションを受信します。
- **UnSubscribeAddress:** 特定のbitcoinアドレスの新しいトランザクションの受信を停止します。

トランザクションは **OnNewTransaction** イベントで受信されます。

```
{
  "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:** 新しいブロックが見つかった際に通知を受け取ります。注意: チェーンが分岐した場合、特定のブロック高さに対して複数の通知が届くことがあります。
- **UnSubscribeBlocks:** 新しいブロックが見つかったときの通知の受信を停止します。注意: チェーンが分岐すると特定のブロック高さに対して複数の通知を受け取る場合があります。

ブロックは **OnNewBlock** イベントで受信されます:

```
{
```



```
"op": "block",
"x": {
  "txIndexes": [
    3187871,
    3187868
  ],
  "nTx": 0,
  "totalBTCsent": 0,
  "estimatedBTCsent": 0,
  "reward": 0,
  "size": 0,
  "blockIndex": 190460,
  "prevBlockIndex": 190457,
  "height": 170359,
  "hash": "00000000000006436073c07dfa188a8fa54fefadf571fd774863cda1b884b90f",
  "mrklRoot": "94e51495e0e8a0c3b78dac1220b2f35ceda8799b0a20cfa68601ed28126cfcc2",
  "version": 1,
  "time": 1331301261,
  "bits": 436942092,
  "nonce": 758889471
}
}
```

API Cex

Cex

WebSocket API は追加のリクエストを送信せずにリアルタイムの通知を取得できるため、取引所からデータを取得するより高速な方法です。

Cex コンポーネントには Cex というプロパティがあり、アカウントデータへのアクセスを取得するために Cex が提供する API キーを入力できます。

メッセージエンコーディング

- すべてのメッセージは JSON 形式でエンコードされています。
- 価格はクライアント側での JSON 解析時の丸め誤差を避けるために文字列として表示されます。
- WebSocket フレームの圧縮はサーバーでサポートされていません。
- 時刻は秒単位の整数 UNIX タイムスタンプとして表されます。

認証

CEX.IO の WebSocket データにアクセスするには、認証が必要です。

- CEX.IO アカウントにログインします。
- <https://cex.io/trade/profile#/api> ページにアクセスしてください。
- 必要な権限のタイプを選択します。
- "Generate Key" ボタンをクリックしてシークレットキーを保存します。アクティベーション後はアクセスできなくなります。
- キーを有効化します。

接続

- 接続されたソケットが 15 秒間非アクティブな場合、CEX.IO サーバーは PING メッセージを送信します。
- PING リクエストのイニシエーターになれるのはサーバーのみです。
- サーバーは認証済みユーザーにのみ ping を送信します。
- ユーザーは PONG メッセージで応答する必要があります。そうでない場合、WebSocket は切断されます。これはライブラリによって自動的に処理されます。
- 認証済みユーザーの場合、15 秒以内にサーバーから通知や ping がない場合は、接続と認証を確保するために「ticker」や「get-balance」などのリクエストを送信してレスポンスを受信することが安全です。

パブリックチャンネル

これらのチャンネルは認証を必要としません。サーバーからのレスポンスは OnCexMessage イベントを通じて受信されます。

- **SubscribeTickers:** すべてのペアで行われた取引価格のみのティッカーフィード（非推奨）

```
{
  "e": "tick",
  "data": {
    "symbol1": "BTC",
    "symbol2": "USD",
    "price": "428.0123"
  }
}
```

- **SubscribeChart:** 始値、高値、安値、終値、出来高を含む OHLCV チャートフィード (非推奨)

```
{
  'e': 'ohlcv24',
  'pair': 'BTC:USD',
  'data': [
    '418.2936',
    '420.277',
    '412.09',
    '416.9778',
    '201451078368'
  ]
}
```

- **ペアのサブスクライブ:** マーケットデプスフィード (非推奨)

```
{
  '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
    }
  }
}
```

- **ペアのサブスクライブ:** 注文板フィード (非推奨)

```
{
  '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
    ]
  }
}
```

プライベートチャンネル

これらのチャンネルにアクセスするには、まず Authenticate メソッドを呼び出します。サーバーからの 応答は OnCexMessage イベントを通じて受信されます。

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 システムでは、注文は取引エンジンまたはアーカイブデータベースに存在できます。注文が完了/キャンセルされたが、まだアーカイブデータベースに移動されていない期間（2 秒以上）が存在する場合があります。つまり、archived-orders/open-orders の呼び出しで表示できません。この呼び出しにより、いかなる場合でも注文情報を取得できます。レスポンスは注文の場所によって異なる形式になる場合があります。

```
{
  "e": "get-order",
  "data": {
    "user": "XXX",
    "type": "buy",
    "symbol1": "BTC",
    "symbol2": "USD",
    "amount": "0.02000000",
    "remains": "0.02000000",
    "price": "50.75",
    "time": 1450214742160,
    "tradingFeeStrategy": "fixedFee",
    "tradingFeeBuy": "5",
    "tradingFeeSell": "5",
    "tradingFeeUserVolumeAmount": "nil",
    "a:USD:c": "1.08",
    "a:USD:s": "1.08",
    "a:USD:d": "0.00",
    "status": "a",
    "orderId": "5582060"
  },
  "oid": "1450214742135_10_get-order",
  "ok": "ok"
}
```

CancelOrderRequest

```
{
  "e": "cancel-order",
  "data": {
    "order_id": "2477098"
    "time": 1443468122895
  },
  "oid": "1435927928274_12_cancel-order",
  "ok": "ok"
}
```

GetArchivedOrders

```
{
  "e": "archived-orders",
  "data": [
    {
      "type": "buy",
      "symbol1": "BTC",
      "symbol2": "USD",
      "amount": 0,
      "amount2": 5000,
      "remains": 0,
      "time": "2015-04-17T10:46:27.971Z",
      "tradingFeeBuy": "2",
      "tradingFeeSell": "2",
      "ta:USD": "49.00",
      "fa:USD": "0.98",
      "orderId": "2340298",
      "status": "d",
    }
  ]
}
```

```

    "a:BTC:cds": "0.18151851",
    "a:USD:cds": "50.00",
    "f:USD:cds": "0.98"
  },
  {
    "type": "buy",
    "symbol1": "BTC",
    "symbol2": "USD",
    "amount": 0,
    "amount2": 10000,
    "remains": 0,
    "time": "2015-04-08T15:46:04.651Z",
    "tradingFeeBuy": "2.99",
    "tradingFeeSell": "2.99",
    "ta:USD": "97.08",
    "fa:USD": "2.91",
    "orderId": "2265315",
    "status": "d",
    "a:BTC:cds": "0.39869578",
    "a:USD:cds": "100.00",
    "f:USD:cds": "2.91"
  }
],
"oid": "1435927928274    15_archived-orders",
"ok": "ok"
}

```

OpenPosition

```

{
  "e": "open-position",
  "oid": "1435927928274_7_open-position",
  "data": {
    'amount': '1',
    'symbol': 'BTC',
    "pair": [
      "BTC",
      "USD"
    ],
    'leverage': '2',
    'ptype': 'long',
    'anySlippage': 'true',
    'eoprice': '650.3232',
    'stopLossPrice': '600.3232'
  }
}

```

GetPosition

```

{
  "e": "get_position",
  "ok": "ok",
  "data": {
    "user": "ud100036721",
    "pair": "BTC:USD",
    "amount": "1.00000000",
    "symbol": "BTC",
    "msymbol": "USD",
    "omamount": "1528.77",
    "lsymbol": "USD",
    "lamount": "3057.53",
    "slamount": "3380.11",
    "leverage": "3",
    "stopLossPrice": "3380.1031",
    "dfl": "3380.10310000",
    "flPrice": "3057.5333333",
    "otime": 1513002370342,
    "psymbol": "BTC",
    "ptype": "long",
    "ofee": "10",
    "pfee": "10",
    "cfee": "10",
    "tfeeAmount": "152.88",
    "rinterval": "14400000",
    "okind": "Manual",
    "a:BTC:c": "1.00000000",
    "a:BTC:s": "1.00000000",

```

```

    "oorder": "89101551",
    "pamount": "1.00000000",
    "lremains": "3057.53",
    "slremains": "3380.11",
    "oprice": "4586.3000",
    "status": "a",
    "id": "125531",
    "a:USD:cds": "4739.18"
  }
}

```

GetOpenPositions

```

{
  'e': 'open_positions',
  "oid": "1435927928256_7_open-positions",
  'ok': 'ok',
  'data': [
    {
      'user': 'ud100036721',
      'id': '104102',
      'otime': 1475602208467,
      'symbol': 'BTC',
      'amount': '1.00000000',
      'leverage': '2',
      'ptype': 'long',
      'psymbol': 'BTC',
      'msymbol': 'USD',
      'lsymbol': 'USD',
      'pair': 'BTC:USD',
      'oprice': '607.5000',
      'stopLossPrice': '520.3232',
      'ofee': '1',
      'pfee': '3',
      'cfee': '4',
      'tfeeAmount': '3.04',
      'pamount': '1.00000000',
      'omamount': '303.75',
      'lamount': '303.75',
      'oorder': '34106774',
      'rinterval': '14400000',
      'dfl': '520.32320000',
      'slamount': '520.33',
      'slremains': '520.33',
      'lremains': '303.75',
      'flPrice': '303.75000000',
      'a:BTC:c': '1.00000000',
      'a:BTC:s': '1.00000000',
      'a:USD:cds': '610.54',
    },
    ...
  ]
}

```

ClosePosition

```

{
  'e': 'close_position',
  "oid": "1435927928364_7_close-position",
  'ok': 'ok',
  'data': {
    'id': 104034,
    'ctime': 1475484981063,
    'ptype': 'long',
    'msymbol': 'USD',
    'pair': {
      'symbol1': 'BTC',
      'symbol2': 'USD'
    },
    'price': '607.1700',
    'profit': '-12.48',
  }
}

```


API Cex Plus

Cex Plus

サポートされている API

- **WebSockets API:** パブリックWebSocketサーバーに接続し、リアルタイムのマーケットデータ更新を提供します。

WebSockets API

WebSocket は TCP ベースの全二重通信プロトコルです。全二重とは、両者が同じ通信チャネルを使用して非同期にメッセージを送り合えることを意味します。このセクションでは、Exchange Plus とクライアントが互いに送り合うべきメッセージを説明します。すべてのメッセージは有効な JSON オブジェクトである必要があります。

WebSocket API は主に REST API で利用できないまたは行いにくい情報を取得したりアクションを実行したりするために使用されます。ただし、一部のリクエストやアクションは REST API と WebSocket API の両方で実行できます。Exchange Plus は、クライアントが以前に送信したリクエストへのレスポンスとして、またはイベントに関する通知（クライアントの事前リクエストなし）としてクライアントにメッセージを送信します。

パブリック API 呼び出し

パブリックAPIのレート制限は、DDoS攻撃からシステムを保護し、すべてのクライアントがExchange Plus APIエンドポイントへの同等の安定したアクセスを確保するために設けられています。パブリックリクエストは、パブリックAPIリクエストが行われるIPアドレスによって制限されます。リクエスト制限は各パブリックAPIコールに関連するコストによって決まります。デフォルトでは、各パブリックリクエストのコストは1ポイントですが、特定のリクエストではこのコストが高くなる場合があります。各メソッドの仕様で最新のリクエストレート制限コスト情報を確認してください。

Exchange Plus は、パブリック API の呼び出しを1分あたり最大100ポイントに制限しています。各パブリック API 呼び出しにはコストがあります（以下を参照）。リクエストのレート制限に達すると、Exchange Plus はエラーで応答し、クライアントに切断イベントを送信してから WebSocket 接続を閉じます。Exchange Plus は次のカレンダー分からクライアントへのサービスを再開します。次の例では、リクエストカウンターは11:02:00.000にリセットされます。

メソッド	説明
GetTicker	このメソッドは、特定の資産の現在の価格、24 時間の価格と出来高の変化、最後の取引イベントなどを含む Ticker の現在情報を取得するように設計されています。
GetOrderBook	このメソッドにより、クライアントは特定の取引ペアの現在の注文板スナップショットを受信できます。
GetCandles	Candlesメソッドを使用すると、クライアントはさまざまな解像度とデータタイプの過去の OHLCV ローソク足を受信できます。クライアントは、追加の時間枠と制限フィルターを指定して、レスポンスをクライアントの要件により正確にすることができます。
GetTradeHistory	このメソッドを使用すると、クライアントは要求された取引ペアで発生した取引の履歴データを取得できます。クライアントは、リクエストに一致する取引を受信するために、期間、取引 ID 範囲、売買方向などの追加フィルターパラメータで取引履歴リクエストを補足できます。
GetServerTime	このメソッドは Exchange Plus サーバーの現在時刻を取得するために使用されます。サーバーの時刻と同期する必要があるアプリケーションに便利です。
GetPairsInfo	Pair Info メソッドにより、クライアントはサポートされているすべての取引ペアのパラメーターを受け取ることができます。

GetCurrenciesInfo	Currencies Info メソッドを使用すると、クライアントは Exchange Plus で設定されたすべての通貨のパラメータ、および Exchange Plus と CEX.IO ウォレット間の入出金の可否を受け取ることができます。
GetProcessingInfo	このリクエストにより、クライアントは外部ウォレットからの入金および外部ウォレットへの出金のオプションに関する詳細情報を取得できます。各サポートされている暗号通貨について、暗号通貨名および入出金可能なブロックチェーンが含まれます。また、各サポートされているブロックチェーンについて、そのブロックチェーン上の暗号通貨の種類、現在の入出金可否、入出金の最低金額、外部出金手数料が記載されています。この情報により、クライアントは入金アドレスの受け取りや外部出金の開始に最適なブロックチェーンを選択する際に、より柔軟な対応が可能になります。
SubscribeOrderBook	クライアントは WebSocket 経由でサブスクライブすることで、リクエストされた取引ペアの注文書フィードをサブスクライブできます。注文書サブスクライブリクエストへのレスポンスとして、クライアントはリクエストされたペアの現在の（初期）注文書スナップショットを指定された seqId 番号とともに受け取ります。注文書の以降の更新を追跡するには、クライアントは WebSocket 経由で「order_book_increment」メッセージをサブスクライブする必要があります。これには取引ペア名、seqId 番号、買い注文と売り注文の価格レベルのデルタが含まれます。
UnSubscribeOrderBook	オーダーブックチャンネルからアンサブスクライブします。
SubscribeTrade	Trade Subscribe メソッドを使用することで、クライアントはリクエストされた取引ペアで発生するトレードイベントのライブフィードを WebSocket 経由で購読できます。Trade Subscribe リクエストへの応答として、クライアントはトレードのサブスクリプションが不要になったときの購読解除に使用するトレードサブスクリプションの一意の識別子を受け取ります。クライアントは「tradeHistorySnapshot」と「tradeUpdate」メッセージを受信するために WebSocket 経由で購読し、リクエストされた取引ペアの初期および定期的なトレード履歴スナップショットとライブトレードイベントを受け取る必要があります。
UnSubscribeTrade	取引チャンネルの購読を解除します。

使用例： BTC-USD ペアの最新ティッカーを取得します。

```
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;
}
```

プライベート API 呼び出し

Exchange Plus はプライベート API へのアクセスを許可するために API キーを使用します。

クライアントは、Exchange Plus Web TerminalのAPI Keys Management Profileセクションを介して、APIキーの生成、設定、管理、権限レベルの設定、APIキーのホワイトリスト登録されたIPなどを行うことができます。

API キーの制限： デフォルトでは、クライアントは最大 5 つの API キーを持てます。

API キーを使用する際に特定の機能へのアクセスを制限するには、各 API キーに対して特定のパーミッションセットを定義する必要があります。定義されたパーミッションセットは必要に応じてさらに編集できます。

API キーに使用可能な権限レベルは以下のとおりです。

- **Read:** アカウント関連データの表示、レポートの受信、市場データのサブスクライブなどの権限レベル。
- **Trade:** アカウントに代わって注文の発注とキャンセルを許可する権限レベル。
- **Funds Internal:** CEX.IO Exchange Plus ポートフォリオのアカウント間（サブアカウント間、またはメインアカウントとサブアカウント間）での資金移動を許可する権限レベル。
- **Funds Wallet:** CEX.IO Exchange Plus ポートフォリオアカウント（メインアカウントとサブアカウント）から CEX.IO ウォレットへ、またその逆に資金を振り替えることができる権限レベル。

メソッド	説明
GetCurrentFee	このメソッドは、クライアントの最新30日間の取引量と曜日（例えば週末は手数料が異なる場合があります）を考慮した特定時点の現在の手数料を示します。
GetFeeStrategy	Fee Strategy は、クライアントの取引量、曜日、ペア、ペアのグループなどを考慮して、クライアントに適用され得るすべての手数料オプションを返します。このメソッドは、クライアントに適用可能なすべての取引手数料の値を含む、一般的な手数料戦略に関する情報を提供します。クライアントの現在の30日間の取引量に基づく現在の取引手数料を受け取るには、クライアントは [Current Fee] メソッドを使用する必要があります。現在の30日間の取引量を受け取るには、クライアントは [Volume] メソッドを使用する必要があります。
GetVolume	このリクエストにより、クライアントは過去 30 日間の USD 相当の取引量を受け取ることができます。
CreateAccount	このリクエストにより、クライアントは新しいサブアカウントを作成できます。デフォルトでは、クライアントはメインアカウントを含めて最大 5 つのサブアカウントを持つことができます。
GetAccountStatus	Account Status V3メソッドを使用することで、クライアントは現在の残高と、変換通貨（デフォルトは「USD」）での参考換算額、各サブアカウントおよび通貨ごとにオープン（アクティブ）注文でロックされている金額を確認できます。クライアントに取引手数料残高が利用可能な場合、レスポンスにはTrading Fee Balanceのプロモ名、通貨名、合計残高、このプロモの有効期限といった一般的な取引手数料残高データも含まれます。サブアカウントで利用可能な取引残高を、現在のサブアカウント残高からサブアカウントのオープン（アクティブ）注文でロックされている残高額を差し引いた値として追跡するのは、クライアントの責任です。
GetOrders	このリクエストにより、クライアントは自分の注文に関する情報を確認できます。
NewOrder	クライアントは、Do My New Order Requestを使用してWebSocket APIを介して新規注文を発注できます。このリクエストへの応答とともに、リクエストが成功するとExchange PlusはAccount EventおよびExecution ReportメッセージをClientに送信します。応答メッセージは、応答送信時点でシステムで利用可能な注文の最新ステータスを示します。ClientがDo My New Order Requestへの応答メッセージを受信しなかった場合、ClientはclientOrderIdパラメータを使用してGet My Orders Requestで注文の現在のステータスを照会できます。新規注文をリクエストする際は、クライアント側の特定の新規注文リクエストに対応するclientOrderIdパラメータを使用することを強く推奨します。Exchange Plusは、同じclientOrderIdで複数回注文を発注することを回避します。同一のclientOrderIdとその他の注文パラメータを持つ複数の新規注文が識別された場合、Exchange Plusは最初の注文のみを発注し、同じパラメータを持つ2番目以降の新規注文リクエストへの応答として、その注文のステータスをClientに返します。同一のclientOrderIdだが他の注文パラメータが異なる複数の新規注文が識別された場合、Exchange Plusは最初の注文のみを処理し、同じclientOrderIdだが他の注文パラメータが異なる2番目以降の新規注文リクエストを拒否します。
NewMarketOrder	新しい成行注文を出します。

NewLimitOrder	新しい指値注文を発注します。
CancelOrder	クライアントは注文をキャンセルできます。このリクエストへのレスポンスとともに、このリクエストが成功した場合、Exchange Plus はクライアントにアカウントイベントと約定レポートメッセージを送信します。また、注文キャンセルのリクエストが拒否された場合、Exchange Plus は注文キャンセル拒否メッセージを送信します。
CancelAllOrders	クライアントは WebSocket API 経由ですべての未決注文をキャンセルできます。このリクエストへのレスポンスとともに、Exchange Plus はすべての未決注文のキャンセルプロセスを開始し、対応するアカウントイベントと約定レポートメッセージをクライアントに送信します。
GetTransactionHistory	このリクエストにより、クライアントは自身の金融取引（入金、出金、内部送金、手数料、または取引）を確認できます。
GetFundingHistory	このリクエストにより、クライアントは入出金トランザクションを確認できます。
InternalTransfer	クライアントはサブアカウント間、またはメインアカウントとサブアカウント間の資金移動をリクエストできます。Exchange Plus はアカウント間の資金移動に手数料を請求しません。このリクエストが成功した場合、Exchange Plus はクライアントにアカウントイベントメッセージを送信します。
GetDepositAddress	このメソッドはクライアントが暗号通貨の入金用の暗号アドレスを受け取るために使用できます。入金アドレスはメインアカウントおよびサブアカウントに対して生成できます。入金アドレスを生成するために利用可能なブロックチェーンの一覧は、クライアントが「処理情報の取得」リクエストを通じて取得できます。
FundsDepositFromWallet	クライアントは、CEX.IO Walletから Exchange Plus アカウントに資金を入金できます。システムは、同じ clientTxId を持つ複数の入金リクエストの処理を回避します。同じ clientTxId を持つ複数の入金リクエストを受信した場合、システムは最初の入金リクエストのみを処理し、同じ clientTxId を持つ 2 番目以降の入金リクエストを拒否します。
FundsWithdrawalToWallet	クライアントは Exchange Plus アカウントから CEX.IO ウォレットに資金を出金できます。システムは同じ clientTxId を持つ複数の出金リクエストを回避します。同一の clientTxId を持つ複数の出金リクエストが受信された場合、システムは最初の出金リクエストのみを処理し、同じ clientTxId を持つ 2 番目以降の出金リクエストを拒否します。
GetWalletBalance	CEX.IO ウォレットの残高情報を取得します。
SubscribeAccountEvents	リアルタイムのアカウントイベント通知（残高変更、注文約定）にサブスクライブします。
UnSubscribeAccountEvents	アカウントイベント通知のサブスクライブを解除します。

例: 注文を取得します。

```
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

注意: API v10以降、DiscordのAPIドメインはdiscordapp.comからdiscord.comに変更されました。コンポーネントはAPIバージョン10と新しいドメインを使用するように更新されました。

ゲートウェイは Discord のセキュアな WebSocket を通じたリアルタイム通信の形式です。クライアントは接続しているゲートウェイを通じてイベントとデータを受信し、REST API を通じてデータを送信します。

認証

まず新しいボットを生成し、API を通じて認証するために使用するボットトークンをコピーします。次に、このトークンを API コンポーネントに設定します。

```
TsgcWSAPI_Discord1->DiscordOptions->BotOptions->Token = "...bot token here...";
```

インテント

特にスケール時に処理が期待されるデータ量を考えると、ステートフルなアプリケーションの維持は困難な場合があります。Gateway Intents は計算負荷を軽減するためのシステムです。

ゲートウェイへの識別時に、Discord によって事前定義された「インテント」（イベントのグループ）に条件付きでサブスクライブできる intents パラメータを指定できます。特定のインテントを指定しない場合、そのグループにまとめられたゲートウェイ イベントは受信されません。有効なインテントは以下の通りです（ゼロ値はすべてのイベントを受信することを意味します）：

```
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
```

ハートビート

ハートビートはコンポーネントによって自動的に処理されるため、気にする必要はありません。クライアントがサーバーに接続すると、サーバーはハートビート間隔とともに HELLO レスポンスを送信し、コンポーネントはレスポンスを読み取り、x秒ごとにピングを送信するようにハートビートを自動調整します。サーバーからクライアントへのピング送信もクライアントによって自動的に処理されます。

接続準備完了

接続の準備ができ、サーバーによるログインと認可が正常に完了すると、**OnDiscordReady** イベントが発生し、サーバーからのアップデートの受信を開始できます。

接続の再開

接続が予期せず閉じられた場合、クライアントが再接続しようとする **OnDiscordBeforeReconnect** イベントが呼び出されます。コンポーネントは成功した再開に必要なすべてのデータを自動的に保存しますが、必要に応じてパラメータを変更できます。再接続せずに新しいクリーンなセッションを開始したい場合は、Reconnect を False に設定してください。

セッションが再開された場合、**OnDiscordResumed** イベントが発生します。新しいセッションの場合は **OnDiscordReady** が発生します。

Dispatch Events

イベントは **OnDiscordDispatch** を通じてディスパッチされるため、ここでサーバーからクライアントに送信されたイベントを読み取ることができます。

```
void OnDiscordDispatch(TObject *Sender, const string aEvent, const string RawData)
{
    DoLog("#discord dispatch: " + aEvent + " " + RawData);
}
```

aEvent パラメーターにはイベント名が含まれます。
RawData には完全なJSONメッセージが含まれます。

HTTP リクエスト

ギルド、ユーザーに関する情報を取得したり、データを更新したりするために、Discordはゲートウェイ WebSocketメッセージの代わりにHTTPリクエストを使用することを要求します。HTTPリクエストを行うために利用可能なすべてのメソッドを以下に示します:

```
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;
```

使用例: 現在のユーザー情報を取得する

```
result = GET_Request("/users/@me");
```

サーバーからのサンプルレスポンス:

```
{
  "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 API は、音声間の会話体験やリアルタイムの文字起こしを含む、低遅延のマルチモーダルインタラクションを実現します。

コンポーネント **TsgcWSAPI_OpenAI** は RealTime OpenAI API を実装します。

設定

method プロパティ を使用して **Conversation** または **Transcription** を選択します。現在は Transcription モードのみがサポートされています。

InputAudio プロパティ では以下のデータをカスタマイズできます:

- **Language:** 例: english (値 = 'en') 。
- **Model:** 使用するモデル。例: whisper-1
- **Prompt:** モデルへの指示を提供するオプションのプロンプト。

OpenAI

OpenAI API は認証に API キーを使用します。リクエストで使用する API キーを取得するには、[API キー](#) ページをご覧ください。

API キーは秘密です！他の人と共有したり、クライアント側のコード（ブラウザー、アプリ）に公開したりしないでください。本番環境のリクエストは、環境変数またはキー管理サービスから API キーを安全に読み込める独自のバックエンドサーバー経由でルーティングする必要があります。

この **API Key** はコンポーネントの **OpenAIOptions.ApiKey** プロパティに設定する必要があります。オプションとして、複数の組織に属するユーザーは、アカウントが組織に属している場合は **OpenAIOptions.Organization** プロパティに組織を設定できます。

API キーが設定されたら、以下に OpenAI API と連携するために利用可能な関数の一覧を示します。

Azure

クライアントは Microsoft Azure OpenAI Services をサポートしているため、Azure アカウントを使用して Azure OpenAI API とやり取りすることもできます。Azure で動作するようクライアントを設定するには、以下の手順に従ってください:

1. プロパティ **OpenAIOptions.Provider** = **oapvAzure** を設定してください。
2. **ResourceName** と **DeploymentId** の値を設定します（これらの値は Azure アカウントで確認できます）
 1. **OpenAIOptions.AzureOptions.ResourceName** = <リソース名>。
 2. **OpenAIOptions.AzureOptions.DeploymentId** = <デプロイメント ID>。
3. Azure アカウントの API キーを設定します
 1. **OpenAIOptions.ApiKey** = <azure api key>。

文字起こし例

openAI API を使用したリアルタイム文字起こしの例を以下に示します。

```
TsgcWebSocketClient *WSCClient = new TsgcWebSocketClient(NULL);
TsgcAudioRecorderWave *oAudio = new TsgcAudioRecorderWave(NULL);
TsgcWSAPI_OpenAI *OpenAI = new TsgcWSAPI_OpenAI(NULL);
OpenAI->Client = WSCClient;
OpenAI->AudioRecorder = oAudio;

OpenAI->OpenAIOptions->APIKey = "your-api-key-here";
OpenAI->OpenAIOptions->Method = sgcoaimTranscription;
OpenAI->OpenAIOptions->Provider = sgcoaipOpenAI;
```

```
OpenAI->InputAudio->Language = "en";
OpenAI->InputAudio->Model = "whisper-1";

void __fastcall TForm1::OnOpenAIAudioTranscriptionCompleted(TObject *Sender, TsgcWSOpenAIConversation_Item_Comple
    Log("#transcription_completed: " + aItem->Transcript);
}
```

API MEXC

MEXC は、セキュアな WebSocket 接続を通じてストリーミングの市場データとアカウントデータを公開するグローバルな暗号通貨取引所です。

WebSocket スポット API は [マーケットストリーム](#) と [ユーザーデータストリーム](#) ドキュメントで公開されている仕様に従います。マーケットチャンネルは Protocol Buffers フレームとして配信され、プライベートストリームは JSON を使用します。**TsgcWSAPI_MEXC** コンポーネントは、接続のライフサイクル、認証、listen キーのメンテナンスを管理するため、アプリケーションはプッシュされたデータの処理に集中できます。

WebSocket によるストリーミングに加えて、ツールキットは完全な Spot HTTP API も公開しています。**TsgcHTTP_API_MEXC_Spot** コンポーネント（ユニット **sgcHTTP_API_MEXC**）は、アプリケーションが市場データを照会したり署名付きリクエストで取引したりできるよう、すべての公式 REST エンドポイントをラップします。

コンポーネントの設定

ベースとなる WebSocket エンドポイントは **wss://wbs-api.mexc.com/ws** です。**MEXCUserDataStreams.UserStream** が有効で、有効な API キーが提供されている場合、コンポーネントは自動的に listen key を要求し、それを接続 URL に付加します。

フォームに **TsgcWebSocketClient** と **TsgcWSAPI_MEXC** コンポーネントをドロップし（またはコードで作成し）、クライアントインスタンスを API コンポーネントに割り当て、クライアントをアクティブにしてセッションを開始します。

```
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");
}
```

プロパティ

- **クライアント: TsgcWebSocketClient** トランSPORTへの参照。コンポーネントはセッションを維持するためにクライアントのハートビート（15秒）を有効にします。
- **MEXCAPI.ApiKey / ApiSecret:** スポットAPI認証情報。キーはプライベートチャンネルに必要です。シークレットはリッスンキーリクエストの署名に必要です。
- **MEXCUserDataStreams.UserStream: True**（デフォルト）の場合、コンポーネントは接続前にリッスンキーを取得し、10分ごとに更新します。パブリックフィードのみで作業する場合は無効にします。
- **MEXCUserDataStreams.ListenKeyOnDisconnect:** リッスンキーがアクティブな状態でWebSocketが切断されたときに実行されるアクション:
 - **blkmxodDeleteListenKey:** RESTを通じてリッスンキーを取り消します（デフォルト）。
 - **blkmxodClearListenKey:** リッスンキーをサーバー側に保持し、キャッシュされた値をクリアします。
 - **blkmxodDoNothing:** 次の接続で同じリッスンキーを再使用します。
- **ListenKey:** REST API が返したアクティブなリッスンキーを公開する読み取り専用プロパティ。
- **RawMessages:** 有効にすると、受信したテキストフレームは解析されずに、継承された WebSocket イベントを通じてプレーン JSON として転送されます。

イベント

- **OnConnect / OnDisconnect**: 下層のWebSocketセッションが確立または終了したときに発生します。
- **OnMEXCMarketStream**: すべての Protobuf マーケットフレームでトリガーされます。ハンドラーは **MessageType** プロパティでペイロードを識別する (Trade、KLine、DiffDepth、LimitDepth、BookTicker、BookTickerBatch、MiniTicker、または MiniTickers) **TsgcMEXCSpotProtoMessage** を受け取ります。デコードされた構造を検査するには **sgcWebSocket_API_MEXC_Proto** のヘルパークラスを使用してください。
- **OnMEXCResponse**: JSON 形式 (**Id**、**Code**、**Message**) でサブスクリプションの確認とping/pong応答を受信します。
- **OnMEXCUserDataStream**: プライベートな口座メッセージ(注文、約定、残高更新)が到着したときに発生します。チャンネル名と生のJSONペイロードが提供されます。
- **OnMEXCException**: REST または WebSocket の例外をキャプチャします (リッスンキーのエラー、HTTP の失敗、解析の問題など)。

次のサンプルは、トレードフレームをデコードする方法を示しています:

WebSocket サブスクリプションメソッド

コンポーネントは各パブリックSpotチャンネルのヘルパーを公開します。 **Subscribe*** メソッドを呼び出してデータのストリーミングを開始し、対応する **UnSubscribe*** で停止します。

パブリックチャンネル

メソッド	パラメータ	説明
SubscribeTrade	Symbol, Interval (ms)	Protobuf publicdeals として配信される集約取引実行。
SubscribeKline	Symbol, Interval	設定可能な間隔 (1分から1か月) のローソク足更新。
SubscribeDiffDepth	Symbol, Interval (ms)	ローカルの板管理のための注文板インクリメンタル深度デルタ。
SubscribeBookDepth	Symbol, Levels	リクエストされた深さ (デフォルト5) での指値板の定期スナップショット。
SubscribeBookTicker	Symbol, Interval (ms)	単一の取引ペアの最良ビッド/アスクの更新。
SubscribeBookTickerBatch	UTC フラグ	複数銘柄の集約ブックティッカー更新。
SubscribeMiniTickers	UTC フラグ	すべてのアクティブな銘柄のローリング 24 時間ミニティッカー統計。
SubscribeMiniTicker	Symbol, UTC flag	単一シンボルのミニティッカー。

プライベートチャンネル

プライベートトピックには有効な API キーが必要です。コンポーネントはアクティブなリッスンキーを使用してサブスクライブし、定期的に更新します。

メソッド	説明
SubscribeAccountUpdate	口座残高とポジションの変更。
SubscribeAccountDeals	約定済み注文の約定レポート。
SubscribeAccountOrders	注文ライフサイクルの更新 (新規、キャンセル、拒否など)。

ストリームを終了するには対応する **UnSubscribe*** メソッドを使用してください。コンポーネントは自動的に **PING** コマンドを送信し、返された **PONG** を検証して接続を監視します。

パブリック REST エンドポイント

公開されている REST メソッド。

メソッド	説明
Ping	/api/v3/ping を呼び出し、取引所が空のオブジェクトで応答したときに True を返す接続チェック。
GetServerTime	/api/v3/time から現在の取引所タイムスタンプを取得します。
GetExchangeInformation	/api/v3/exchangeInfo から取引ルールとシンボルメタデータを返します。
GetOrderBook	/api/v3/depth からオーダーブックのスナップショットをダウンロードします。オプションの limit パラメーターでレベル数を制御します（デフォルト 100）。
GetTrades	オプションの limit （デフォルト 100）付きで /api/v3/trades から最近のパブリック取引を取得します。
GetHistoricalTrades	オプションの limit と fromId フィルターを使用した過去の取引データを提供します。
GetAggregateTrades	/api/v3/aggTrades を介して圧縮/集約された取引を返します。オプションの limit 、 fromId 、 startTime 、 endTime フィルターをサポートします。
GetKlines	/api/v3/klines から時間範囲と件数フィルターをサポートするローソク足データをダウンロードします。
GetAveragePrice	/api/v3/avgPrice から現在の加重平均価格を返します。
Get24hrTicker	/api/v3/ticker/24hr を介して 1 つまたはすべてのシンボルの 24 時間ティッカー統計を取得します。
GetPriceTicker	/api/v3/ticker/price を使用して最新の価格を取得します。すべての取引ペアの価格を取得するには空のシンボルを渡してください。
GetBookTicker	単一シンボルまたは全取引所リクエストに対応する /api/v3/ticker/bookTicker から最良の買い/売り気配値を取得します。

プライベート REST エンドポイント

プライベートエンドポイントには有効な API キーと署名されたクエリ文字列が必要です。コンポーネントはタイムスタンプ、recvWindow、HMAC 署名を自動的に追加します。

メソッド	説明
GetAccountInformation	/api/v3/account からの残高と権限を返します。
GetOpenOrders	現在のオープン注文を一覧表示します（オプションでシンボルでフィルタリング）。
GetAllOrders	オプションの時間制限とフィルターを含む過去の注文を取得します。
GetOrder	orderId または origClientOrderId のいずれかを指定して、特定の注文のステータスをクエリします。
GetMyTrades	オプションの limit 、 fromId 、 startTime 、 endTime フィルターとともに、プライベートな取引約定を一覧表示します。
NewOrder	/api/v3/order に注文（マーケット、リミット、ストップなど）を発注します。オプションで timeInForce 、 quantity 、 quoteOrderQty 、 price

	、 newClientId 、 stopPrice 、 icebergQty 、および追加パラメーターが指定できます。
TestNewOrder	NewOrder と同じパラメーターセットを受け入れる /api/v3/order/test に検証のみのリクエストを送信します。
CancelOrder	/api/v3/order の特定の注文を orderId または origClientId でキャンセルします。
CancelAllOrders	シンボルのすべてのオープン注文を /api/v3/openOrders 経由で一括キャンセルします。
GetSubAccounts	管理下のサブアカウントを一覧表示します (/api/v3/sub-account/list)。
GetSubAccountAssets	特定のサブアカウントの残高を返します。
TransferSubAccount	/api/v3/sub-account/transfer を通じてサブアカウント間で資産を振り替えます。金額と、オプションで振替 type を指定します。
GetDepositAddress	オプションの network フィルターを使用してデポジットアドレスをリクエストします。
GetDepositHistory	オプションの coin 、 status 、 startTime 、 endTime フィルターを使用して入金記録を取得します。
GetWithdrawHistory	コイン 、 ステータス 、およびオプションの時間範囲でフィルタリングされた出金履歴を返します。
出金	必須の coin 、 address 、 amount に加えて、オプションの network 、 addressTag 、 withdrawOrderId および追加パラメータを含めて、 /api/v3/capital/withdraw/apply 経由で出金リクエストを送信します。
BatchOrders	1つのリクエストで複数のシンボルのバッチ注文を送信します。
GetTradeFee	特定のシンボルの取引手数料率を取得します。
GetDepositHistory	オプションの coin 、 status 、 startTime 、 endTime フィルターを使用して入金記録を取得します。
CancelWithdraw	出金 ID によって保留中の出金リクエストをキャンセルします。
CreateInternalTransfer	アカウント間で資産を内部転送します。
GetTransferHistory	アカウント間の内部振替の履歴を取得します。

API MEXC Futures

MEXC の永久および配信契約は、公式の [Futures WebSocket 仕様](#) に記載された専用ストリーミング API を公開しています。TsgcWSAPI_MEXC_Futures コンポーネントは、リアルタイムデリバティブデータを消費するために必要な接続、ログインハンドシェイク、トピック管理をカプセル化します。

MEXC はデリバティブ取引用の REST インターフェイスも維持しています。sgcHTTP_API_MEXC に含まれる TsgcHTTP_API_MEXC_Futures コンポーネントは、公式エンドポイントと 1 対 1 に対応する HTTP ヘルパーで WebSocket フィードを補完します。

コンポーネントの設定

Futures WebSocket エンドポイントは `wss://contract.mexc.com/edge` です。メッセージは JSON オブジェクトとしてエンコードされ、取引所はコンポーネントによって自動的に処理される定期的な **ping/pong** フレームを要求します。

プライベート通知（注文およびアカウントデータ）を受け取るには、認証が必要です。WebSocket クライアントを有効にする前に **MEXCAPI.ApiKey** と **MEXCAPI.ApiSecret** を設定してください。認証情報が存在する場合、コンポーネントはソケット接続後すぐにログインリクエスト（HMAC SHA256）に署名します。

```
TsgcWebSocketClient *WSClient = new TsgcWebSocketClient();
TsgcWSAPI_MEXC_Futures *Futures = new TsgcWSAPI_MEXC_Futures();
Futures->Client = WSClient;
Futures->MEXCAPI->ApiKey = "YOUR_KEY";
Futures->MEXCAPI->ApiSecret = "YOUR_SECRET";
if (WSClient->Connect()) {
    Futures->SubscribeDepth("BTC_USDT", true);
}
```

プロパティ

- **クライアント**：API コンポーネントが使用する WebSocket トランスポート。ソケットをアクティブに保つためにハートビートサポートが有効になっています。
- **MEXCAPI.ApiKey / ApiSecret**：ログインリクエストへの署名に必要です。認証情報がない場合、パブリックチャンネルのみアクセスできます。
- **RawMessages**：有効にすると JSON パーサーをバイパスし、生のテキストフレームが基本イベントを通じて公開されます。

イベント

- **OnConnect / OnDisconnect**：WebSocket ライフサイクル通知。
- **OnMEXCLogin**：認証ログインが成功した後に発生します。
- **OnMEXCSubscribed / OnMEXCUnsubscribed**：サブスクリプション変更の確認。イベントは MEXC が返したチャンネル名を公開します。
- **OnMEXCMessage**：チャンネル識別子とともにすべての市場またはアカウントの更新を配信します。
- **OnMEXCError**：サーバーが `rs.error` ペイロード（無効なパラメータ、認証失敗など）で応答した場合にトリガーされます。
- **OnMEXCException**：JSON データの処理やネットワーク操作の実行中に発生したローカル例外を報告します。

次のハンドラーは、サブスクリプションの確認を記録し、受信した板情報スナップショットを出力します：

WebSocket サブスクリプションメソッド

各ヘルパーは公式 API で説明された **sub/unsub** リクエストをラップします。ストリームをキャンセルするには対応する **UnSubscribe*** メソッドを使用してください。

メソッド	パラメータ	説明
SubscribeDeal	銘柄	契約で実行された取引（チャンネル: deal ）。
SubscribeTickers	-	すべての契約のグローバルなティッカー統計です。
SubscribeTicker	銘柄	単一銘柄のティッカーサマリー。
SubscribeDepth	Symbol, Compress	増分オーダーブックの更新
SubscribeDepthFull	Symbol, Level	設定可能なレベル数（デフォルト 20）での完全な深さスナップショット。
SubscribeKline	Symbol, Interval	サポートされているタイムフレームのローソク足データ（Min1、Min5、Min15、Min30、Min60、Hour4、Hour12、Day1、Week1、Month1）。
SubscribeFundingRate	銘柄	最新のファンディングレートのお知らせ。
SubscribeIndexPrice	銘柄	原資産の価格ストリーム。
SubscribeFairPrice	銘柄	取引所がプッシュするマーク（フェア）価格の更新。

サーバーがリクエストを確認すると、チャンネル名（例: **rs.sub.depth.BTC_USDT**）とともに **OnMEXCSubscribed** イベントが発生します。エラーはトラブルシューティング用のサーバーメッセージを含む **OnMEXCError** を通じて転送されます。

プライベートチャンネル

プライベートチャンネルには有効なAPIキーが必要です。資格情報が提供されると、コンポーネントは自動的に認証します。

メソッド	説明
SubscribePersonalOrder	個人の注文更新。
SubscribePersonalOrderDeal	個人注文の約定（フィル）更新。
SubscribePersonalPosition	個人のポジション更新。
SubscribePersonalPlanOrder	個人プランの（トリガー）注文のアップデート。
SubscribePersonalStopOrder	個人の逆指値注文の更新。
SubscribePersonalStopPlanOrder	個人の停止プラン注文の更新。
SubscribePersonalRiskLimit	個人リスク制限の更新。
SubscribePersonalADLLevel	個人 ADL (自動デレバレッジ) レベルの更新。
SubscribePersonalAsset	個人資産の更新。

パブリック REST エンドポイント

公開されている REST メソッド。

メソッド	説明
GetPing	/api/v1/ping で API の到達可能性を確認します。
GetServerTime	/api/v1/time からサーバーのタイムスタンプを返します。
GetContracts	利用可能なコントラクトのリストを提供します (/api/v1/contract/detail) 。

GetDepth	/api/v1/contract/depth から板情報をダウンロードします。オプションの limit パラメータ（デフォルト 50）で価格レベルの数を選択できます。
GetDeals	/api/v1/contract/deals を使用して、オプションの limit （デフォルト 100）付きで最近の取引を取得します。
GetKlines	オプションの startTime 、 endTime 、 limit フィルター（デフォルト 200）付きで /api/v1/contract/kline を通じてローソク足データを返します。
GetIndexPrice	要求したシンボルの基礎となるインデックス価格を取得します（ /api/v1/contract/indexPrice ）。
GetFairPrice	/api/v1/contract/fairPrice からコントラクトの公正（マーク）価格を返します。
GetFundingRate	/api/v1/contract/fundingRate を使用して、シンボルの最新のファンディングレートを報告します。
GetAllTickers	リアルタイム価格サマリーを含むすべての先物コントラクトのティッカーを取得します。
GetFundingRateHistory	コントラクトの過去のファンディングレートレコードを取得します。
GetFairPriceKline	契約の公正（マーク）価格のローソク足データを取得します。
GetIndexPriceKline	契約のインデックス価格Kline/ローソク足データを取得します。

プライベート REST エンドポイント

プライベートデリバティブエンドポイントには API 認証情報が必要です。コンポーネントはタイムスタンプと `recvWindow` を使用して各リクエストに自動的に署名します。

メソッド	説明
GetAccountAssets	/api/v1/private/account/assets からマージン残高を返します。
GetPositionList	/api/v1/private/position/list を通じて現在のポジションを一覧表示します。オプションでシンボルによってフィルタリングできます。
SetPositionLeverage	シンボルのレバレッジと、指定されている場合はマージンモードを更新します。
PlaceOrder	/api/v1/private/order で先物注文を発注します。 symbol 、 side 、 positionSide 、 type 、 quantity を指定し、オプションで price 、 clientOrderId 、追加パラメータを指定します。
CancelOrder	/api/v1/private/order/cancel を使用して、 orderId または clientOrderId で特定の注文をキャンセルします。
CancelAllOrders	/api/v1/private/order/cancel-all を通じてシンボルのすべてのオープン注文をキャンセルします。
GetOpenOrders	オプションのシンボルフィルター付きで /api/v1/private/order/list/open を通じて現在のオープン注文を一覧表示します。
GetOrderHistory	オプションの銘柄、時間範囲、件数パラメータを指定して注文履歴を取得します（ /api/v1/private/order/list/history ）。
GetFundingHistory	/api/v1/private/account/funding から過去のfunding支払いを、オプションの startTime 、 endTime 、 limit フィルターとともに返します。
GetOpenPositions	現在開いているすべてのポジションを取得します。
ChangeMargin	ポジションのマージン額を変更します。
GetPositionMode	現在のポジションモード（一方向モードまたはヘッジモード）を取得します。
ChangePositionMode	一方向モードとヘッジモード間でポジションモードを変更します。
PlaceBatchOrder	単一のバッチリクエストで複数の先物注文を発注します。
GetOrderDetail	特定の注文の詳細情報を取得します。

GetOrderDealDetails	特定の注文の約定/取引の詳細を取得します。
PlaceTriggerOrder	条件が満たされたときに実行されるトリガー（プラン）注文を発注します。
CancelTriggerOrder	特定のトリガー注文をキャンセルします。
CancelAllTriggerOrders	すべてのオープントリガー注文をキャンセルします。
GetTriggerOrders	トリガー（プラン）注文のリストを取得します。
GetStopOrders	ストップ注文の一覧を取得します。
CancelStopOrder	特定のストップ注文をキャンセルします。
CancelAllStopOrders	すべてのオープンストップ注文をキャンセルします。

API Bitget

Bitget

サポートされている API

- **WebSocket API:** WebSocket サーバーに接続し、リアルタイムの市場データ更新、アカウント変更などを提供します。
- **REST API:** HTTPリクエストを送信して、市場データの取得、注文の発注、アカウントデータの取得などを行います。

プロパティ

チャンネルタイプを選択するには **BitgetChannel** プロパティを使用してください:

- **bgcSpot:** Spot WebSocket API に接続します。
- **bgcFutures:** Futures WebSocket API に接続します。

Bitget プロパティで以下のプロパティを設定できます。

- **ApiKey:** Bitget アカウントで新しい API キーをリクエストし、その値をこのプロパティにコピーしてください。ApiKey が設定されている場合、クライアントは WebSocket プライベートサーバーに接続します。空の場合は WebSocket パブリックサーバーに接続します。
- **ApiSecret:** API のシークレット値です。
- **Passphrase:** 新しい API キーを作成するときに定義するカスタム文字列です。

接続

クライアントがBitgetサーバーに正常に接続すると、**OnConnect** イベントが発生します。**OnConnect** イベントが発生した後、Bitgetサーバーとのメッセージの送信および受信を開始できます。プライベートWebSocketチャンネルに接続する場合は、いずれかのチャンネルにサブスクライブする前に、**OnBitgetAuthentication** イベントが発生するまで待機し、successパラメータがtrueかどうかを確認する必要があります。

```
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");
}
```

イベント

Bitget クライアントは、接続フローを制御し WebSocket サーバーから送信されたデータを取得するための以下のイベントを実装しています。

- **OnConnect:** WebSocket クライアントが WebSocket サーバーに接続したときに発生します。
- **OnDisconnect:** WebSocket クライアントが WebSocket サーバーから切断されたときに発生します。
- **OnBitgetAuthentication:** クライアントがプライベート WebSocket サーバーに対して認証を行ったときに発生します。
- **OnBitgetSubscribe:** クライアントが WebSocket チャンネルをサブスクライブしたとき。
- **OnBitgetUnSubscribe:** クライアントが WebSocket チャンネルのサブスクリプションを解除したとき。
- **OnBitgetData:** クライアントがサーバーからデータを受信したとき。
- **OnBitgetError:** Bitget WebSocket 接続中にエラーが発生したとき。
- **OnBitgetHTTPException:** REST リクエスト中にエラーが発生した場合。

WebSocket API

WebSocket フィードは注文と取引のリアルタイムの市場データ更新を提供します。
以下のチャンネルにサブスクライブできます:

メソッド	パブリック またはプ ライベート	説明
SubscribeTicker	パブリック	インストルメントのティックーストリームにサブスクライブします。
SubscribeTrade	パブリック	最新の取引ストリームにサブスクライブします。
SubscribeCandle	パブリック	ローソク足ストリームにサブスクライブします。複数の時間間隔をサポートします (例: candle1m、candle5m、candle1H)。
SubscribeOrderBook	パブリック	オーダーブックストリームにサブスクライブします。異なる深度レベルをサポートします。
SubscribeOrders	プ ライ ベ ー ト	注文の更新をサブスクライブします。認証が必要です。
SubscribePositions	プ ライ ベ ー ト	ポジション更新をサブスクライブします。認証が必要です。
SubscribeAccount	プ ライ ベ ー ト	アカウント残高の更新をサブスクライブします。認証が必要です。

認証成功後にプライベート WebSocket チャンネルをサブスクライブする例を以下に示します。

```
void OnBitgetAuthentication(TObject *Sender, bool aSuccess, const string aError, const string aRawMessage)
{
    if (aSuccess == true)
    {
        oBitget->SubscribeOrders("BTCUSD");
        oBitget->SubscribeAccount();
    }
}
```

REST API

REST API は市場とプライベートアカウントからデータをリクエストするためのパブリックとプライベートのメソッドを提供します。コンポーネントの **REST_API** プロパティを通じて REST API にアクセスします。

メソッド	パブリック / プ ライ ベ ー ト	説明
GetServerTime	パブリック	サーバー時刻を取得します。
GetTickers	パブリック	1 つまたはすべてのシンボルのティックー情報を取得します。
GetOrderBook	パブリック	シンボルの注文書を取得します。
GetCandles	パブリック	シンボルのローソク足/Kline データを取得します。
GetRecentTrades	パブリック	銘柄の直近の取引を取得します。
PlaceOrder	プ ライ ベ ー ト	新しい注文を発注します。
CancelOrder	プ ライ ベ ー ト	既存の注文をキャンセルします。
GetOpenOrders	プ ライ ベ ー ト	未決注文のリストを取得します。
GetOrderDetail	プ ライ ベ ー ト	特定の注文の詳細を取得します。
GetOrderHistory	プ ライ ベ ー ト	注文履歴を取得します。

GetAccountAssets	プライベート	口座資産情報を取得します。
GetAccountInfo	プライベート	アカウント情報を取得します。

以下に REST API の使用例を示します。

```
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

サポートされている API

- **WebSocket API:** WebSocket サーバーに接続し、リアルタイムの市場データ更新、アカウント変更などを提供します。
- **REST API:** HTTPリクエストを送信して、市場データの取得、注文の発注、アカウントデータの取得などを行います。

プロパティ

GateIOChannel プロパティを使用してチャンネルタイプを選択します：

- **giocSpot:** Spot WebSocket APIに接続します。
- **giocFutures:** 先物 WebSocket API に接続します。

GateIO プロパティで以下のプロパティを設定できます。

- **ApiKey:** Gate.ioアカウントで新しいapiキーをリクエストできます。値をこのプロパティにコピーするだけです。APIKeyが設定されている場合、クライアントはプライベートサーバーに対して認証します。空の場合、パブリックチャンネルのみが利用可能になります。
- **ApiSecret:** API のシークレット値です。

接続

クライアントがGate.ioサーバーに正常に接続すると、**OnConnect** イベントが発生します。**OnConnect** イベントが発生した後、Gate.ioサーバーとのメッセージの送信および受信を開始できます。プライベートWebSocketチャンネルに接続する場合は、プライベートチャンネルにサブスクライブする前に、**OnGateIOAuthentication** イベントが発生するまで待機し、successパラメータがtrueかどうかを確認する必要があります。

```
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");
}
```

イベント

GateIO クライアントは、接続フローを制御して WebSocket サーバーから送信されたデータを取得するために次のイベントを実装します：

- **OnConnect:** WebSocket クライアントが WebSocket サーバーに接続したときに発生します。
- **OnDisconnect:** WebSocket クライアントが WebSocket サーバーから切断されたときに発生します。
- **OnGateIOAuthentication:** クライアントがプライベートWebSocketサーバーに対して認証するときに発生します。
- **OnGateIOSubscribe:** クライアントが WebSocket チャンネルをサブスクライブしたとき。
- **OnGateIOUnSubscribe:** クライアントが WebSocket チャンネルからサブスクライブ解除したとき。
- **OnGateIOData:** クライアントがサーバーからデータを受信したとき。
- **OnGateIOError:** GateIO WebSocket 接続中にエラーが発生した場合。
- **OnGateIOHTTPException:** RESTリクエスト中にエラーが発生した場合。

WebSocket API

WebSocket フィードは注文と取引のリアルタイムの市場データ更新を提供します。以下のチャンネルにサブスクライブできます：

メソッド	パブリック またはプ ライベート	説明
SubscribeTicker	パブリック	銘柄のティックーストリームをサブスクライブします。
SubscribeTrades	パブリック	最新の取引ストリームにサブスクライブします。
SubscribeCandlesticks	パブリック	ローソク足ストリームをサブスクライブします。複数のインターバル（例: 1m、5m、15m、30m、1h）に対応しています。
SubscribeOrderBook	パブリック	注文板ストリームをサブスクライブします。設定可能な深さレベルと更新間隔をサポートします。
SubscribeOrders	プ ライ ベ ー ト	注文の更新をサブスクライブします。認証が必要です。
SubscribeBalances	プ ライ ベ ー ト	残高更新をサブスクライブします。認証が必要です。

パブリック WebSocket チャンネルのサブスクライブ例を以下に示します。

```
void OnConnect(TsgcWSConnection *Connection)
{
    oGateIO->SubscribeTicker("BTC_USDT");
    oGateIO->SubscribeOrderBook("BTC_USDT", "20", "100ms");
}
```

REST API

REST API は市場とプライベートアカウントからデータをリクエストするためのパブリックとプライベートのメソッドを提供します。コンポーネントの **REST_API** プロパティを通じて REST API にアクセスします。

メソッド	パブリック / プ ライ ベ ー ト	説明
GetCurrencyPairs	パブリック	利用可能な通貨ペアの情報を取得します。
GetTickers	パブリック	1 つまたはすべての通貨ペアのティックー情報を取得します。
GetOrderBook	パブリック	通貨ペアの注文板を取得します。
GetCandlesticks	パブリック	通貨ペアのローソク足/K ラインデータを取得します。
GetTrades	パブリック	通貨ペアの最新取引を取得します。
PlaceOrder	プ ライ ベ ー ト	新規注文を発注します。指値注文と成行注文をサポートします。
CancelOrder	プ ライ ベ ー ト	既存の注文をキャンセルします。
GetOrder	プ ライ ベ ー ト	特定の注文の詳細を取得します。
GetOpenOrders	プ ライ ベ ー ト	未決注文のリストを取得します。
GetSpotAccounts	プ ライ ベ ー ト	スポットアカウント情報を取得します。

以下に REST API の使用例を示します。

```
TsgcWS_API_GateIO *oGateIO = new TsgcWS_API_GateIO();
oGateIO->GateIO->ApiKey = "your_api_key";
oGateIO->GateIO->ApiSecret = "your_api_secret";
// Get tickers
oGateIO->REST_API->GetTickers("BTC_USDT");
// Place an order
oGateIO->REST_API->PlaceOrder("BTC_USDT", "buy", "0.001", "50000", "limit");
```


API Deribit

Deribit

Deribit は、Bitcoin と Ethereum の先物およびオプション取引を提供する暗号通貨デリバティブ取引所です。

サポートされている API

- **WebSocket API:** WebSocket サーバーに接続し、リアルタイムの市場データ更新、アカウント変更などを提供します。
- **REST API:** HTTP リクエストを送信して、市場データの取得、注文の発注、アカウントデータの取得などを行います。

プロパティ

Deribit プロパティで次のプロパティを設定できます。

- **ApiKey:** Deribit アカウントで新しい API キーをリクエストし、その値をこのプロパティにコピーできます。ApiKey が設定されている場合、クライアントはプライベートサーバーに対して認証します。空の場合、パブリックチャンネルのみが利用可能です。
- **ApiSecret:** API のシークレット値です。
- **TestNet:** 有効にすると、Deribit TestNet 環境に接続します（デフォルトは無効）。実際の資金を危険にさらさずにテストするのに役立ちます。

接続

クライアントがDeribitサーバーに正常に接続すると、**OnConnect** イベントが発生します。**OnConnect** イベントが発生した後、Deribitサーバーとのメッセージの送信および受信を開始できます。プライベートWebSocketチャンネルに接続する場合は、プライベートチャンネルにサブスクライブする前に、**OnDeribitAuthentication** イベントが発生するまで待機し、successパラメータがtrueかどうかを確認する必要があります。

```
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");
}
```

イベント

Deribit クライアントは、接続フローを制御し WebSocket サーバーから送信されるデータを取得するために、以下のイベントを実装しています。

- **OnConnect:** WebSocket クライアントが WebSocket サーバーに接続したときに発生します。
- **OnDisconnect:** WebSocket クライアントが WebSocket サーバーから切断されたときに発生します。
- **OnDeribitAuthentication:** クライアントがプライベート WebSocket サーバーに対して認証を行ったときに発生します。
- **OnDeribitSubscribe:** クライアントが WebSocket チャンネルにサブスクライブしたとき。
- **OnDeribitUnSubscribe:** クライアントが WebSocket チャンネルからサブスクライブ解除したとき。
- **OnDeribitData:** クライアントがサーバーからデータを受信したとき。
- **OnDeribitError:** Deribit WebSocket 接続中にエラーが発生した場合。
- **OnDeribitHTTPException:** REST リクエスト中にエラーが発生した場合。

WebSocket API

WebSocket フィードは注文と取引のリアルタイムの市場データ更新を提供します。次の**Public channels**にサブスクライブできます:

メソッド	説明
SubscribeTicker	商品のtickerストリームにサブスクライブします。リアルタイムの価格と取引高のデータを提供します。
SubscribeTrades	インストゥルメントの最近の取引ストリームにサブスクライブします。
SubscribeOrderBook	オーダーブックストリームにサブスクライブします。グループ化、深さレベル、更新間隔の設定をサポートします。
SubscribeCandle	ローソク足ストリームを購読します。複数の解像度（例：1、3、5、10、15、30、60、120、180、360、720、1D）をサポートします。
SubscribePerpetual	ある銘柄の永続契約データにサブスクライブします。ファンディングレートおよびその他の永続特有の情報を提供します。
SubscribeBookChange	銘柄の板変更通知にサブスクライブします。

以下のプライベートチャンネル（認証が必要）をサブスクライブできます：

メソッド	説明
SubscribeOrders	銘柄の注文アップデートを購読します。
SubscribePositions	ポジション更新をサブスクライブします。通貨や銘柄の種類によるフィルタリングに対応しています。
SubscribeAccountChanges	通貨のアカウント残高変化をサブスクライブします。
SubscribeUserTrades	ある銘柄のユーザー取引更新をサブスクライブします。

以下に WebSocket チャンネルへのサブスクライブの例を示します。

```
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 API は市場、取引、アカウント、ウォレットからデータをリクエストするためのパブリックおよびプライベートメソッドを提供します。コンポーネントの **REST_API** プロパティを通じて REST API にアクセスします。
マーケットデータ（パブリック）

メソッド	説明
GetInstruments	通貨で利用可能なインストゥルメントを取得します。
GetTicker	インストゥルメントのティックー情報を取得します。
GetOrderBook	インストゥルメントの注文板を取得します。
GetTrades	銘柄の最新の取引を取得します。
GetCurrencies	サポートされている通貨のリストを取得します。
GetIndexPrice	インデックス名の現在のインデックス価格を取得します。
GetFundingRateHistory	銘柄のファンディングレート履歴を取得します。
GetFundingRateValue	インストゥルメントの現在のファンディングレート値を取得します。
GetBookSummaryByCurrency	通貨のすべての銘柄の書籍の概要を取得します。
GetBookSummaryByInstrument	特定の銘柄の板サマリーを取得します。

取引 (プライベート)

メソッド	説明
Buy	買い注文を発注します。マーケット注文と指値注文タイプをサポートします。
売り	売り注文を発注します。成行注文と指値注文をサポートします。

CancelOrder	注文 ID で既存の注文をキャンセルします。
CancelAllOrders	すべての未決注文をキャンセルします。
CancelAllByInstrument	特定の銘柄のすべてのオープン注文をキャンセルします。
EditOrder	既存の注文を編集します（数量または価格を変更）。
GetOpenOrders	通貨のオープン注文の一覧を取得します。
GetOrderState	特定の注文のステータスを取得します。
GetOrderHistory	通貨の注文履歴を取得します。

Account (Private)

メソッド	説明
GetPositions	通貨のポジションを取得します。
GetAccountSummary	通貨のアカウントサマリーを取得します。
GetSubAccounts	サブアカウントの一覧を取得します。
GetTransactionLog	通貨のトランザクションログを取得します。

ウォレット（プライベート）

メソッド	説明
GetDeposits	通貨の入金履歴を取得します。
GetWithdrawals	通貨の出金履歴を取得します。
GetTransfers	通貨の送金履歴を取得します。

以下に REST API の使用例を示します。

```
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](#)

サポートされている API

- **WebSocket API:** WebSocket サーバーに接続し、リアルタイムの市場データ更新、アカウント変更などを提供します。
- **REST API:** HTTPリクエストを送信して、市場データの取得、注文の発注、アカウントデータの取得などを行います。

プロパティ

Channel プロパティを使用してチャンネルタイプを選択します:

- **cccMarket:** 公開市場データ用のマーケット WebSocket API に接続します。
- **cccUser:** プライベートアカウントデータ用のユーザー WebSocket API に接続します（認証が必要です）。

CryptoCom プロパティで次のプロパティを構成できます。

- **ApiKey:** Crypto.com アカウントで新しい API キーをリクエストし、その値をこのプロパティにコピーしてください。プライベートチャンネルと認証済み REST API 呼び出しに必須です。
- **ApiSecret:** API のシークレット値です。

接続

クライアントがCrypto.comサーバーに正常に接続すると、**OnConnect** イベントが発生します。**OnConnect** イベントが発生した後、Crypto.comサーバーとのメッセージの送信および受信を開始できます。Userチャンネルに接続する場合は、プライベートチャンネルにサブスクライブする前に、**OnCryptoComAuthentication** イベントが発生するまで待機し、successパラメータがtrueかどうかを確認する必要があります。

```
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");
}
```

イベント

Crypto.com クライアントは、接続フローを制御し、WebSocket サーバーから送信されたデータを取得するために次のイベントを実装しています。

- **OnConnect:** WebSocket クライアントが WebSocket サーバーに接続したときに発生します。
- **OnDisconnect:** WebSocket クライアントが WebSocket サーバーから切断されたときに発生します。
- **OnCryptoComAuthentication:** クライアントがプライベート WebSocket サーバーに認証したときに発火します。
- **OnCryptoComSubscribe:** クライアントが WebSocket チャンネルをサブスクライブしたとき。
- **OnCryptoComUnSubscribe:** クライアントが WebSocket チャンネルからアンサブスクライブしたとき。
- **OnCryptoComData:** クライアントがサーバーからデータを受信したとき。
- **OnCryptoComError:** Crypto.com WebSocket 接続中にエラーが発生した場合。
- **OnCryptoComHTTPException:** REST リクエスト中にエラーが発生した場合。

WebSocket API

WebSocket フィードは注文と取引のリアルタイムの市場データ更新を提供します。以下のマーケットチャンネル（公開）を購読できます。

メソッド	説明
SubscribeTicker	商品のtickerストリームにサブスクライブします。リアルタイムの価格と取引高のデータを提供します。
SubscribeTrade	インストゥルメントの最近の取引ストリームにサブスクライブします。
SubscribeCandlestick	ローソク足ストリームをサブスクライブします。複数の間隔をサポートしています（例: 5m、15m、30m、1h、4h、1D）。
SubscribeBook	注文板ストリームにサブスクライブします。設定可能な深度レベル（例: 10、50）をサポートします。

以下のユーザーチャンネル（プライベート、認証が必要）を購読できます。

メソッド	説明
SubscribeOrders	注文の更新をサブスクライブします。オプションで銘柄名でフィルタリングできます。
SubscribeUserTrades	ユーザー取引のアップデートを購読します。オプションでインストゥルメント名でフィルタリングできます。
SubscribeBalance	アカウントの残高更新にサブスクライブします。

マーケットチャンネルへの購読例を以下に示します。

```
void OnConnect(TsgcWSConnection *Connection)
{
    oCryptoCom->SubscribeTicker("BTC_USDT");
    oCryptoCom->SubscribeBook("BTC_USDT", "10");
}
```

REST API

REST API は市場とプライベートアカウントからデータをリクエストするためのパブリックとプライベートのメソッドを提供します。コンポーネントの **REST_API** プロパティを通じて REST API にアクセスします。

メソッド	パブリック / プライベート	説明
GetInstruments	パブリック	利用可能な銘柄のリストを取得します。
GetTicker	パブリック	1 つまたはすべての銘柄のティッカー情報を取得します。
GetOrderBook	パブリック	インストゥルメントの注文板を取得します。
GetTrades	パブリック	銘柄の最新の取引を取得します。
GetCandlestick	パブリック	銘柄のローソク足/ライン データを取得します。
CreateOrder	プライベート	新しい注文を作成します。指値注文と成行注文をサポートします。
CancelOrder	プライベート	既存の注文をキャンセルします。
CancelAllOrders	プライベート	インストゥルメントのすべてのオープン注文をキャンセルします。
GetOpenOrders	プライベート	未決注文のリストを取得します。
GetOrderDetail	プライベート	特定の注文の詳細を取得します。
GetOrderHistory	プライベート	注文履歴を取得します。
GetAccountSummary	プライベート	アカウントのサマリー情報を取得します。

以下に REST API の使用例を示します。

```
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（旧Huobi）は、国際的な多言語暗号通貨取引所です。HTX APIコンポーネントは、既存のHuobi WebSocket APIを補完するHTTP REST APIを提供します。

サポートされている API

- **WebSocket API:** WebSocket サーバーに接続し、リアルタイムの市場データの更新、アカウントの変更などを提供します。WebSocket サブスクリプションメソッドについては Huobi API ドキュメントをご参照ください。
- **REST API:** HTTPリクエストを送信して、市場データの取得、注文の発注、アカウントデータの取得などを行います。

プロパティ

Huobi プロパティ（WebSocket の場合）または HTXOptions プロパティ（REST API の場合）で次のプロパティを設定できます。

- **ApiKey:** HTX アカウントで新しい API キーをリクエストし、その値をこのプロパティにコピーしてください。ApiKey が設定されている場合、クライアントは WebSocket プライベートサーバーに接続します。空の場合は WebSocket パブリックサーバーに接続します。
- **ApiSecret:** API のシークレット値です。

接続

HTX コンポーネントは Huobi WebSocket API クライアントを拡張します。クライアントが HTX サーバーへの接続に成功すると、**OnConnect** イベントが発生します。ApiKey が空でない場合、クライアントはプライベート WebSocket サーバーへの接続を試み、プライベートメソッドのみが利用可能になります。ApiKey が空の場合、クライアントはパブリック WebSocket サーバーに接続し、パブリックメソッドのみが利用可能になります。

```
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");
}
```

イベント

HTX クライアントは、接続フローを制御し、WebSocket サーバーから送信されたデータを取得するために次のイベントを実装しています：

- **OnConnect:** WebSocket クライアントが WebSocket サーバーに接続したときに発生します。
- **OnDisconnect:** WebSocket クライアントが WebSocket サーバーから切断されたときに発生します。
- **OnHuobiSubscribed:** 購読が成功した後に呼び出されるイベント。
- **OnHuobiUnSubscribed:** アンサブスクライブが成功した後に呼び出されるイベント。
- **OnHuobiUpdate:** データ（kline、market depth など）に更新があるたびにこのイベントが呼び出されます。
- **OnHuobiError:** HTX APIにエラーがある場合、このイベントがエラーに関する情報を提供します。

WebSocket API

WebSocket APIはリアルタイムのマーケットデータ更新を提供します。パブリックチャンネル（SubscribeKLine、SubscribeMarketDepth、SubscribeTradeDetail など）とプライベートチャンネル（SubscribeOrderUpdates、SubscribeTradeClearing、SubscribeAccountChange）を含むWebSocketサブスクリプションメソッドの完全なリストについては、Huobi APIのドキュメントを参照してください。

REST API

REST API は市場とプライベートアカウントのデータをリクエストするためのパブリックメソッドとプライベートメソッドを提供します。REST API は **TsgcHTTP_API_HTX** コンポーネントを通じてアクセスされます。
 マーケットデータ (パブリック)

メソッド	説明
GetServerTime	サーバー時刻を取得します。
GetSymbols	利用可能な取引シンボルの一覧を取得します。
GetCurrencys	サポートされている通貨のリストを取得します。
GetMarketTickers	すべての取引シンボルのティックャーを取得します。
GetMarketDetail	シンボルの 24 時間のマーケット詳細を取得します。
GetMarketDepth	シンボルのオーダーブック深度を取得します。設定可能な深度タイプと深度レベルをサポートします。
GetMarketTrade	銘柄の最新の取引を取得します。
GetMarketHistoryTrade	シンボルの最新取引履歴を取得します。
GetMarketHistoryKline	ローソク足/Klineデータを取得します。複数の期間 (1min、5min、15min、30min、60min、4hour、1day、1mon、1week、1year) に対応しています。

取引 (プライベート)

メソッド	説明
PlaceOrder	新しい注文を発注します。買い成行、売り成行、買い指値、売り指値、その他の注文タイプをサポートします。
CancelOrder	注文 ID で既存の注文をキャンセルします。
GetOrder	特定の注文の詳細を取得します。
GetOpenOrders	アカウントのオープンオーダーリストを取得します。
GetOrderHistory	シンボルの注文履歴を取得します。

Account (Private)

メソッド	説明
GetAccounts	アカウントのリストを取得します。
GetAccountBalance	特定のアカウントの残高を取得します。
GetAccountAssetValuation	アカウントタイプ (例: スポット) の総資産評価を取得します。

以下に REST API の使用例を示します。

```
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 Platformのクラウドホスト版を使用して、メッセージを**送信**および**受信**します。**Cloud API**を使用すると、独自のサーバーをホストするコストをかけずにWhatsApp Business APIを実装でき、ビジネスメッセージングをより簡単にスケールさせることもできます。Cloud APIは、送受信合わせて毎秒最大80メッセージ（テキストおよびメディアメッセージを含む）をサポートします。

WhatsApp Business API を使用すると、中規模および大規模の企業が顧客と大規模にコミュニケーションできます。APIを使用して、企業は何千もの顧客とエージェントまたはボットを接続するシステムを構築でき、プログラムによる通信と手動での通信の両方を可能にします。さらに、CRMやマーケティングプラットフォームなど多数のバックエンドシステムとAPIを統合できます。

機能

企業は Cloud API でより速く新機能を利用できます。現在、WhatsApp Business Cloud API には WhatsApp Business API で利用可能なすべての機能が含まれています。

WhatsApp Cloud APIの便利な機能:

- **CRM、アナリティクス、サードパーティアプリなどのツールと WhatsApp メッセージングを統合**します
- **グリーンチェック、確認済みの WhatsApp Business プロフィール。**
- **WhatsApp ブロードキャストと一括メッセージング**
- アプリまたはインターフェイスなし。BSP または CRM 経由で使用。
- **WhatsApp チャットボット** とサードパーティアプリを使用したチャット **自動化**
- **大規模な WhatsApp メッセージをスケジュール**します。
- **インタラクティブメッセージング機能**にはリストメッセージ、返信ボタン、CTAメッセージが含まれます。

最も一般的な使用方法

- **設定**
 - [WhatsApp アプリの作成](#)
 - [WhatsApp 電話番号 ID](#)
 - [WhatsApp トークン](#)
 - [WhatsApp Webhook](#)
 - [WhatsApp セキュリティ](#)
- **メッセージ**
 - [WhatsApp メッセージ送信](#)
 - [WhatsApp インタラクティブメッセージ送信](#)
 - [WhatsApp テンプレートメッセージ送信](#)
 - [WhatsApp メッセージとステータス通知の受信](#)
 - [WhatsApp ファイル送信](#)
 - [WhatsApp メディアダウンロード](#)

はじめに

テスト番号を使用して最初のメッセージを送受信するには、次の手順を実行します。

1. 開発者アセットとプラットフォームアクセスの設定

- [Meta デベロッパーとして登録](#)する
- [アカウントの二要素認証を有効にする](#)

- **Meta アプリを作成する:** developers.facebook.com > **My Apps** > **Create App** へ移動します。「Business」タイプを選択し、画面の指示に従ってください。

アプリダッシュボードから WhatsApp に接続するアプリをクリックします。下にスクロールして "WhatsApp" 製品を見つけて **セットアップ** をクリックします。

次の画面では、既存のビジネスマネージャーを選択するオプション（お持ちの場合）、または必要であればオンボーディングプロセスで自動的に作成することができます（後でビジネスをカスタマイズできます）。選択して **Continue** をクリックしてください。

Continue をクリックすると、オンボーディングプロセスは次のアクションを実行します:

- アプリは選択したビジネスマネージャーまたは自動的に作成されたビジネスマネージャーに関連付けられています。
- WhatsApp のテスト用電話番号がビジネスに追加されます。このテスト用電話番号を使用すると、実際の電話番号を登録または移行することなく、WhatsApp Business Platform を試すことができます。テスト用電話番号は、最大5人の受信者（世界中のどこにいてもかまいません）に対して無制限にメッセージを送信できます。

2. テストメッセージの送信

IDE を開いて新しいプロジェクトを作成してください。TsgcWhatsapp_Client コンポーネントをドロップして以下のプロパティを設定してください:

- **WhatsappOptions.PhoneNumberId:** メッセージの送信に使用される電話番号のID。
- **WhatsappOptions.Token:** 24 時間有効な一時アクセストークンです。

これら 2 つのプロパティが適切に設定されたら、**SendTest** メソッドを呼び出して **WhatsApp Business Platform** を使用して電話番号に**最初のメッセージ**を送信します。

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendTest("34605889421");
```

3. Webhook の設定

メッセージを受信したとき、またはメッセージのステータスが変更されたときに通知を受けるには、アプリの Webhooks エンドポイントを設定する必要があります。Webhooks の設定は電話番号のステータスには影響せず、メッセージの送受信を妨げません。

開始するには、まずエンドポイントを作成する必要があります。WhatsApp クライアントコンポーネントの **ServerOptions** プロパティを設定して以下のプロパティを設定します:

- **ServerOptions:** バインドする IP アドレス、リッスンポート、SSL 使用の有無（WebHook はセキュアサーバーで実行する必要があります。SSL を使用するようにサーバーを設定するか、WebHook リクエストを HTTPS 以外のサーバーにプロキシできます）を設定できます。サーバーは **TsgcWebsocketHTTPServer** をベースとしています。
 - **WebhookOptions:** このプロパティを使用して、後で facebook 開発者アカウントに設定する Webhook プロパティを設定できます。
 - **Endpoint:** エンドポイントの名前です。デフォルトは /webhook です。例: サーバーが <https://www.esegece.com> でリッスンしている場合、エンドポイントは "<https://www.esegece.com/webhook>" になります
 - **Token:** あなたが定義した任意の値を含むセキュリティ文字列です。Webhook 登録が正しいことを確認するために使用されます。

サーバーを構成した後、**StartServer** メソッドを使用してサーバーを起動し、受信リクエストを受け付けることができます。

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->ServerOptions->WebhookOptions->PhoneNumberId = "/webhook";
```

```
oClient->ServerOptions->WebhookOptions->Token = "MySecretToken";
oClient->StartServer();
```

エンドポイントの準備ができれば、アプリのダッシュボードに移動します。

App Dashboardで、WhatsApp製品を見つけて **Configuration** をクリックします。次にwebhooksセクションを見つけて **Configure a webhook** をクリックします。クリック後、画面にダイアログが表示され、2つの項目を尋ねられます：

- コールバック URL：これは Meta がイベントを送信する URL です。
- Verify Token：この文字列はウェブフックエンドポイントを作成する際に設定します。

情報を追加した後、**確認して保存** をクリックします。

App Dashboardに戻り、左側のパネルで **WhatsApp > Configuration** をクリックします。Webhooksの下で、**Manage** をクリックします。通知を受け取ることができるすべてのオブジェクトが表示されたダイアログボックスが開きます。ユーザーからメッセージを受信するには、**messages** の **Subscribe** をクリックします。

4. テストメッセージを受信します。

新しいメッセージを受信するたびに、クライアントイベント **OnMessageReceived** が呼び出されます。

```
void OnMessageReceived(TObject *Sender, TsgcWhatsapp_Receive_Message *aMessage, ref bool aMarkAsRead)
{
    DoLog("Received: " + aMessage->Messages->_Message[0]->Id);
}
```

Webhookの設定が完了したら、使用したテスト番号にメッセージを送信してください。メッセージの内容が記載されたWebhooks通知がすぐに届くはずですが、

WhatsApp API では、過去 24 時間以内に連絡してきていない電話番号に自由形式のテキストメッセージを送信することはできません。開発者アカウント番号に一度もテキストを送ったことのない電話番号に自由形式のテキストメッセージを送信する唯一の方法は、Meta によって事前に承認されたテンプレートを送ることです。自由形式のテキストメッセージのテストでこの制限を回避するには、まず宛先番号から開発者アカウント番号に WhatsApp メッセージを送信してください。その後、24 時間は自由形式のテキストメッセージを送信できます。

イベント

OnBeforeSendMessage

このイベントは、メッセージがWhatsAppサーバーに送信される前に呼び出されます。RawMessageパラメータを通じて内部メッセージにアクセスできます。

OnBeforeSubscribe

このイベントはサーバーがトピックをサブスクライブする前に呼び出されます。Accept パラメータを使用してサブスクリプションを許可または拒否します。デフォルトでは、サーバーはリクエストされたすべてのイベントをサブスクライブします。

OnRawMessage

このイベントはサーバーがまだ解析されていない新しいメッセージを受信したときに呼び出されるため、生のメッセージにアクセスできます。

OnMessageReceived

このイベントはサーバーが新しいメッセージを受信して解析した後に呼び出されます。MarkAsRead パラメータを True に設定すると、送信者はダブルチェックを受け取ります。

OnMessageSent

このイベントは、サーバーが以前に送信したメッセージに関する新しいステータスメッセージを受信するたびに呼び出されます。Status プロパティを読み取り、メッセージが送信済み、配信済み、または既読かどうかを確認します。

WhatsApp アプリの作成

developers.facebook.com にアクセスしてアプリを作成します。

アプリの種類としてビジネスタイプを選択して続行します。

The screenshot shows the 'Create an App' interface. On the left, there's a sidebar with 'Type' (selected) and 'Details'. The main area is titled 'Select an app type' with a note: 'The app type can't be changed after your app is created. [Learn more](#)'. There are three options: 'Business' (selected with a radio button), 'Consumer', and 'None'. Each option has a brief description. At the bottom right, there are 'Previous' and 'Next' buttons.

アプリの名前を指定します ("WhatsApp" や "Facebook" などの商標名は使用しないでください)。

The screenshot shows the 'Create an App' interface, Step 2: 'Provide basic information'. The sidebar shows 'Type' (checked) and 'Details'. The main area has three input fields: 'Display name' (with a note: 'This is the app name associated with your app ID. You can change this later.'), 'App contact email' (with a note: '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.'), and 'Business Account - Optional' (with a note: 'To access certain permissions or features, apps need to be connected to a Business Account.' and a dropdown menu showing 'No Business Manager account selected'). At the bottom, there's a checkbox for 'By proceeding, you agree to the [Facebook Platform Terms](#) and [Developer Policies](#)'. There are 'Previous' and 'Create app' buttons.

アプリが作成されたら、次の画面で **WhatsApp ボタン** をクリックしてアプリに WhatsApp 送信機能を追加します。



次の画面では、WhatsAppアプリをFacebookビジネスアカウントにリンクする必要があります。まだアカウントをお持ちでない場合は、新しいビジネスアカウントを作成するオプションもあります。

WhatsApp 電話番号 ID

WhatsApp Cloud APIに登録すると、Facebookは、アプリケーションのデフォルトの送信アドレスとなるTest WhatsApp電話番号を提供します。受信者については、開発フェーズ中に支払いを行うことなく、最大5つの電話番号を追加するオプションがあります。

後で独自の電話番号を登録することで、5件の電話番号という制限を回避できます。

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();  
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
```

WhatsApp トークン

WhatsApp Cloud API は、Cloud API を使用してメッセージを送信するために有効なトークンが必要です。

Facebook はテスト用の WhatsApp 電話番号を提供しており、最大 5 つの電話番号にメッセージを送信できます。後で独自の電話番号を登録することでこの制限を解除できます。

WhatsApp API は23時間有効な**一時アクセストークン**を提供します。このトークンはメッセージを送信できるようにするために TsgcWhatsApp_Client コンポーネントに設定する必要があります。

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();  
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
```

長期間有効なトークンが必要な場合は、システムユーザーを作成(または更新)し、**whatsapp_business_messaging**権限で新しいトークンを生成できます。これにより、23時間ごとにトークンを更新することなく、WhatsAppメッセージを送受信できます。

WhatsApp Webhook

ビジネスが受け取るメッセージや顧客プロフィールの更新に関する通知を受け取るには、Webhook をサブスクライブしてください。

エンドポイントの作成

通知の受信を開始する前に、通知を受信するサーバー上にエンドポイントを作成する必要があります。

エンドポイントは 2 種類の HTTPS リクエストを処理する必要があります: 確認リクエストとイベント通知です。どちらのリクエストも HTTPS を使用するため、サーバーには有効な TLS または SSL 証明書が正しく設定およびインストールされている必要があります。自己署名証明書はサポートされていません。

WhatsApp 設定で Webhook を設定する際は、サーバーがリッスンしているエンドポイントと、Webhook エンドポイントの登録時に使用されるトークン（任意の値）を定義する必要があります。このトークンはサブスクライバーが有効であることを確認するために使用されます。

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->ServerOptions->WebhookOptions->PhoneNumberId = "/webhook";
oClient->ServerOptions->WebhookOptions->Token = "MySecretToken";
oClient->StartServer();
```

Webhook が設定されたら、新しいメッセージが受信されるたびに通知を受け取るために **Messages Webhook** フィールドにサブスクライブします。

[SSL サーバー](#)の設定についてはこちらをご覧ください。

WhatsApp セキュリティ

新しいメッセージが受信されるたびに、またはメッセージの新しいステータスが発生するたびに、サーバーは [Webhook](#) で設定されたエンドポイントに通知を受信します。リクエストがWhatsApp Cloud APIサーバーからのものであることを確認するため、リクエストには署名を含むヘッダーが含まれています。WhatsAppクライアントを設定して、メッセージを処理する前に署名を検証できます。

これを行うには、まず **ServerOptions.Application.Secret** プロパティにアプリケーションシークレットを設定し、**VerifySignature** プロパティを有効にする必要があります。

設定が完了すると、新しいメッセージを受信するたびに署名が検証され、誤っている場合はエラー500を返し、メッセージは処理されません。

WhatsApp メッセージの送信

すべてのAPI呼び出しはAccess Tokenで認証される必要があります。開発者は、**App Dashboard > WhatsApp > Getting Started**で生成されたアクセストークンを使用してAPI呼び出しを認証できます

API 呼び出しはメッセージ ID を文字列として返します。

Text Messages

SendMessageText メソッドを呼び出し、次のパラメータを渡します。

- **aTo:** 電話番号
- **aText:** メッセージのテキスト。

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendMessageText("34605889421", "Hello from sgcWebSockets!!!");
```

Image Messages

SendMessageImage メソッドを呼び出し、以下のパラメータを渡します:

- **aTo:** 電話番号
- **aLink:** 送信する画像がある URL
- **aCaption:** 画像のタイトル（オプション）。

```
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");
```

ドキュメントメッセージ

SendMessageDocument メソッドを呼び出して次のパラメーターを渡してください:

- **aTo:** 電話番号
- **aLink:** 送信するドキュメントが存在するURL
- **aCaption:** ドキュメントのタイトル（オプション）。
- **aFileName:** ファイル名（オプション）。

```
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");
```

オーディオメッセージ

メソッド **SendMessageAudio** を呼び出し、以下のパラメータを渡します:

- **aTo:** 電話番号
- **aLink:** 送信するオーディオの 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");
```

ビデオメッセージ

SendMessageVideo メソッドを呼び出し、次のパラメータを渡します:

- **aTo:** 電話番号
- **aLink:** 送信するビデオがある 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");
```

スタッカーメッセージ

メソッド **SendMessageSticker** を呼び出し、以下のパラメーターを渡してください:

- **aTo:** 電話番号
- **aLink:** 送信するステッカーがある URL

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendMessageSticker("34605889421", "https://www.stickers.com/sticker");
```

位置情報メッセージ

SendMessageLocation メソッドを呼び出し、次のパラメータを渡してください:

- **aTo:** 電話番号
- **aLongitude:** 位置の経度。
- **aLatitude:** 場所の緯度。
- **aName:** 場所の名前。
- **aAddress:** 場所のアドレス。aName が設定されている場合のみ表示されます。

```
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

メソッド**SendMessageContact** を呼び出し、次のパラメータを渡します:

- **aTo:** 電話番号
- **aName:** 通常表示されるフルネーム（必須）。
- **aPhone:** 電話番号（オプション）。
- **aEmail:** メール（オプション）。

```
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 Send Interactive Messages

インタラクティブメッセージは、ユーザーが WhatsApp 上でビジネスから求めるものを簡単に見つけて選択できるようにします。テスト中、インタラクティブメッセージ機能を使用したチャットボットは、テキストベースのものと比較して応答率とコンバージョンが大幅に向上しました。

以下のメッセージはインタラクティブとみなされます：

- **リストメッセージ**: 最大 10 個のオプションのメニューを含むメッセージ。このタイプのメッセージは、ビジネスとのインタラクションにおいてユーザーが選択を行うためのよりシンプルで一貫した方法を提供します。
- **返信ボタン**: 最大3つのオプションを含むメッセージ（各オプションはボタン）。このタイプのメッセージは、ビジネスとやり取りする際にユーザーがメニューから選択する際のより迅速な方法を提供します。返信ボタンは、ボタン付きのインタラクティブテンプレートと同じユーザーエクスペリエンスを持ちます。

インタラクティブメッセージ仕様

- インタラクティブなメッセージは同じフロー内で組み合わせることができます。
- ユーザーはリストまたはボタンメッセージから同時に複数のオプションを選択できませんが、以前のメッセージに戻って再度開くことができます。
- リストまたは返信ボタンメッセージは通知として使用できません。現在、ユーザーが最後にメッセージを送信してから 24 時間以内にのみ送信できます。24 時間ウィンドウ外でメッセージを送信しようとするエラーメッセージが返されます。

使用すべき場面

リストメッセージは以下のような複数のオプションを提示するのに最適です：

- カスタマーサービスまたは FAQ メニュー
- テイクアウトメニュー
- 近くの店舗や場所の選択
- 利用可能な予約時間
- 繰り返す最近の注文を選択する

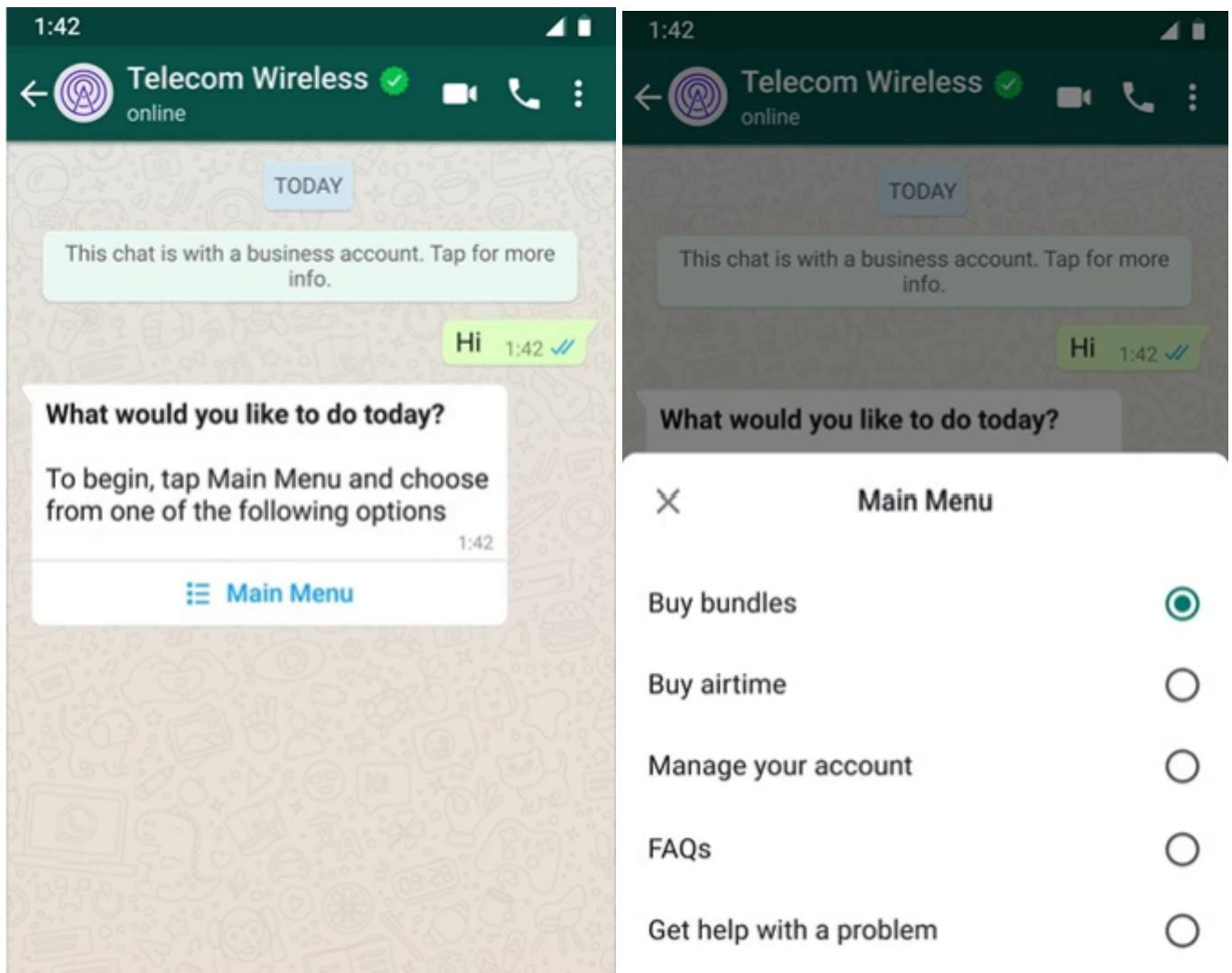
返信ボタンは、以下のような限られたオプションからの素早い返信に最適です：

- エアタイムリチャージ
- 個人情報の変更
- 前の注文の再注文
- 返品のリクエスト
- フード注文にオプションの追加品目を加える
- 支払い方法の選択

返信ボタンは、一般的な応答では不十分な「パーソナライズされた」ユースケースに特に役立ちます。

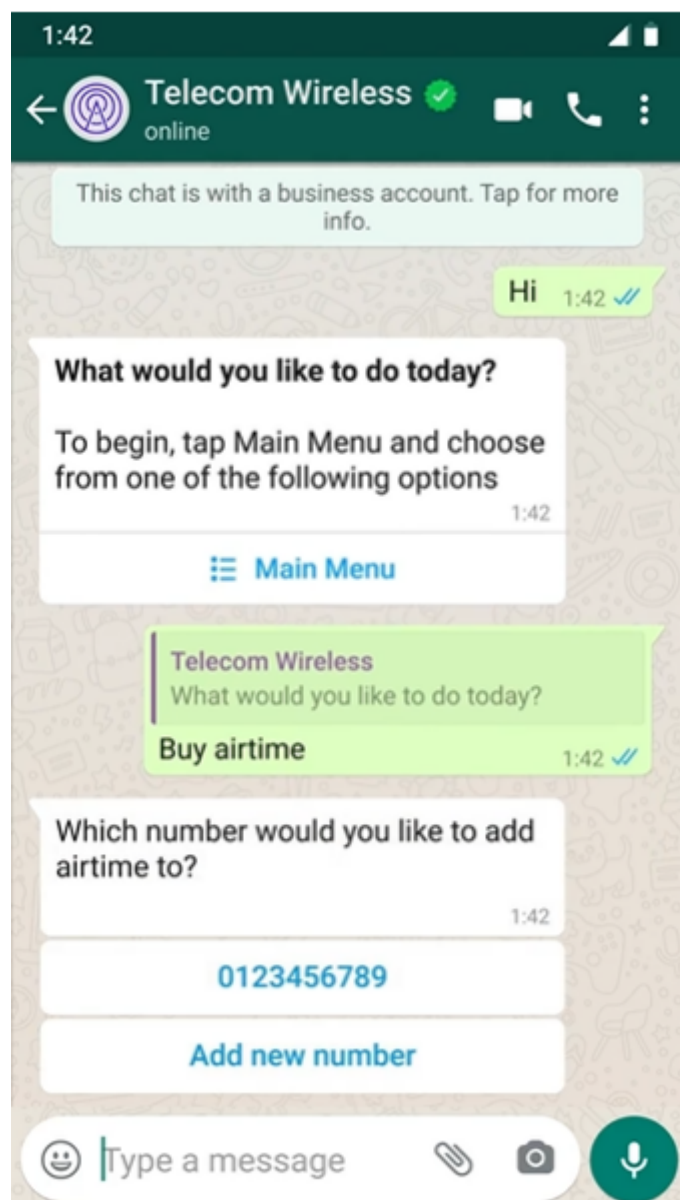
インタラクティブリスト

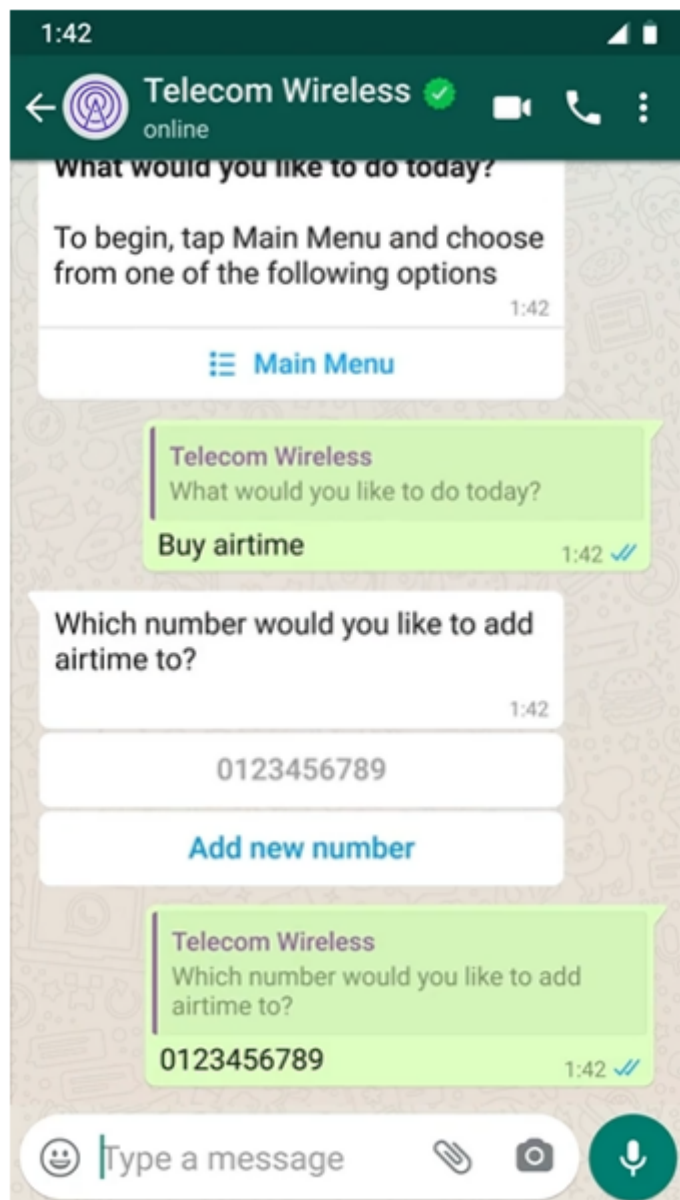
```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendMessageInteractiveList("
34605889421
", "What Would you like to do today?", "To begin, Tap Main Menu and choose from of the following options", "", "M
```



返信ボタン

```
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 テンプレートメッセージの送信

メソッド **SendMessageTemplate** を呼び出し、次のパラメーターを渡します:

- **aTo:** 電話番号
- **aTemplate:** テンプレートの識別子。
- **aLanguageCode:** テンプレートの言語。

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendMessageTemplate("34605889421", "hello_world", "en_US");
```

テンプレートメッセージパラメーター

テンプレートにはパラメータを含めることができます。以下にパラメータを含むデフォルトテンプレートの例を示します。

```
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;
}
```

テンプレートメッセージのアップロード画像

画像へのリンクを使用する代わりに、まず画像をサーバーにアップロードしてからドキュメントの ID を設定するテンプレートの例を以下に示します。

```
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 メッセージの受信とステータス通知

ビジネスが受信したメッセージとカスタマープロフィールの更新に関する通知を受け取るには [Webhook](#) にサブスクライブします。

トリガーイベントが発生するたびに、WhatsApp Business Platform はそのイベントを検出し、事前に指定した Webhook URL に通知を送信します。2 種類の通知を受け取ることができます:

- **受信したメッセージ:** このアラートはメッセージを受信したことを通知します。
- **メッセージのステータスと料金の通知:** このアラートは、メッセージのステータスが変化したとき(たとえば、メッセージが既読になった、または配信された場合)を通知します。

受信メッセージ

新しいメッセージが受信されるたびに **OnMessageReceived** イベントが呼び出され、メッセージのコンテンツにアクセスしてメッセージを既読としてマークできます。

以下に例を示します。新しいテキストメッセージが受信されると、送信者にエコーされます。

```
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;
            }
        }
    }
}
```

送信済みメッセージ

WhatsApp Business Platform はユーザーとの間のメッセージステータスを通知します。メッセージが正常に送信されると、メッセージが送信、配信、既読になったときに通知を受け取ります。アプリ内でのこれらの通知の順序は、メッセージステータスの実際のタイミングを反映しない場合があります。必要に応じてタイムスタンプを確認してタイミングを判断してください。

- **sent:** ビジネスがユーザー主導の会話の一部としてメッセージを送信したときに受信される通知です (その会話が無料エントリーポイントから始まっていない場合) :
- **delivered :** ビジネスのメッセージが配信され、そのメッセージがユーザー主導の会話の一部である場合 (その会話が無料エントリーポイントで開始されていない場合)、次の通知が受信されます。
- **read:** ユーザーがメッセージを読んだときに、次の通知を受信します。

新しいステータスを受信するたびに、**OnMessageSent** イベントが呼び出されます。

```
void OnWhatsAppMessageSent(TObject *Sender, const TsgcWhatsApp_Receive_Message *aMessage, TsgcWhatsAppSendMessage *aStatus)
{
    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 ファイル送信

すべてのAPI呼び出しはAccess Tokenで認証される必要があります。開発者は、**App Dashboard > WhatsApp > Getting Started**で生成されたアクセストークンを使用してAPI呼び出しを認証できます

API 呼び出しはメッセージ ID を文字列として返します。

WhatsApp APIを使用してファイルを送信する場合、まずメッセージがWhatsAppサーバーにアップロードされ、その後、ファイルのアップロード後に返されたオブジェクトIDを使用して新しいメッセージが送信されます。

Image Messages

SendMessageImage メソッドを呼び出し、以下のパラメータを渡します:

- **aTo:** 電話番号
- **aFileName:** 送信する画像ファイルの完全なファイル名（パスを含む）。
- **aFileType:**
 - image/jpeg
 - image/png
- **aCaption:** 画像のタイトル（オプション）。

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendFileImage("34605889421", "c:\\images\\image.png", "image/png");
```

ドキュメントメッセージ

SendMessageDocument メソッドを呼び出して次のパラメーターを渡してください:

- **aTo:** 電話番号
- **aFileName:** 送信するドキュメントファイルのフルファイル名（パス含む）。
- **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:** ドキュメントのタイトル（オプション）。

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->SendFileDocument("34605889421", "c:\\MyDocuments\\invoice.pdf", "application/pdf");
```

オーディオメッセージ

メソッド **SendMessageAudio** を呼び出し、以下のパラメータを渡します:

- **aTo:** 電話番号

- **aFileName:** 送信するオーディオファイルのフルファイル名（パス含む）。
- **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");
```

ビデオメッセージ

SendMessageVideo メソッドを呼び出し、次のパラメータを渡します:

- **aTo:** 電話番号
- **aFileName:** 送信するビデオファイルの完全なファイル名（パスを含む）。
- **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");
```

スタッカーメッセージ

メソッド **SendMessageSticker** を呼び出し、以下のパラメーターを渡してください:

- **aTo:** 電話番号
- **aFileName:** 送信するステッカーファイルのフルパスを含むファイル名。
- **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

メディアファイルリンクを含むメッセージを受け取った場合、**DownloadMedia** メソッドを使用してメディアファイルをダウンロードできます。

```
TsgcWhatsapp_Client oClient = new TsgcWhatsapp_Client();
oClient->WhatsappOptions->PhoneNumberId = "107809351952205";
oClient->WhatsappOptions->Token = "EAA040pgZAs98BAGj3nCFGr...ZB2t8mmLB2LRXJkte2Y5PMNh2";
oClient->DownloadMedia("38923878928822", "c:\\whatsapp\\media\\image.png");
```

以前にアップロードされたメディアファイルを削除するには、**DeleteMedia** を呼び出し、オブジェクト ID を引数として渡してください。

API Telegram

Telegram

Telegram は 2 種類の API を提供しています。**Bot API** はボットを作成し HTTPS をプロトコルとして使用するプログラムを作成できます。**Telegram API** と **TDLib** はカスタム Telegram クライアントを構築でき、Bot API よりはるかに強力です。

sgcWebSockets は tdjson ライブラリを通じて **TDLib** をサポートしています。これにより、独自の Telegram クライアントを構築できます。TDLib はネットワーク実装の詳細、暗号化、ローカルデータストレージをすべて処理します。TDLib はすべての Telegram 機能をサポートしています。

TDLib (Telegram Database Library) の利点

- **クロスプラットフォーム:** Windows、Android、iOS、MacOS、Linux などで使用できます。
- **使いやすさ:** アプリケーションと Telegram 間の通信に JSON メッセージを使用します。
- **高パフォーマンス:** Telegram Bot API では、各 TDLib インスタンスが 24,000 以上のボットを処理します。
- **Consistent:** TDLib はすべての更新が正しい順序で配信されることを保証します。
- **Reliable:** TDLib は、低速で信頼性の低いインターネット接続でも安定しています。
- **Secure:** すべてのローカルデータは、ユーザー提供の暗号化キーを使用して暗号化されます。
- **完全非同期:** TDLib へのリクエストはお互いをブロックしません。レスポンスは利用可能になったときに送信されます。

設定

Windows

TDLib は他のサードパーティライブラリを必要とします: OpenSSL と ZLib。これらのライブラリは tdjson ライブラリと一緒にデプロイする必要があります。

* Windows バージョンは VCRuntime が必要です。Microsoft からダウンロードできます: <https://www.microsoft.com/en-us/download/details.aspx?id=52685>。インストール後も問題が続く場合は、アプリケーションと同じフォルダーに次の DLL をコピーしてみてください: VCRUNTIME140.dll および VCRUNTIME140_1.dll。

次のライブラリをアプリケーションと同じディレクトリにコピーしてください:

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

ライブラリ libtdjson.dylib をデバイスにデプロイすると、SetTDJsonPath を使用してライブラリの場所を設定できます。例:

"Contents\MacOS\" にデプロイする場合は、TPath.GetDirectoryName(ParamStr(0)) フォルダーにパスを設定する必要があります。

OSXARM64

-

ライブラリ libtdjson.dylib をデバイスにデプロイすると、SetTDJsonPathを使用してライブラリの場所を設定できます。例:

"Contents\MacOS\" にデプロイする場合は、TPath.GetDirectoryName(ParamStr(0)) フォルダーにパスを設定する必要があります。

Linux64

デバイスに libtdjson.so ライブラリをデプロイし、SetTDJsonPath メソッドを呼び出してライブラリパスを設定します。

Android

デバイスにライブラリ libtdjsonandroid.so をデプロイします。例: Android64 ライブラリをデプロイする場合は、Project/Deployment の RemotePath を "library\lib\arm64-v8a\" に設定します。Android32 の場合は、RemotePath を "library\lib\armeabi-v7a\" に設定します。

iOS64

ライブラリ libtdjson.a を次のディレクトリにコピーしてください:

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

Telegram アプリケーションの作成

Telegram API を使用して独自のアプリケーションを開発するための API id を取得するには、以下の手順を実行する必要があります。

- 任意のアプリケーションを使用して Telegram にサインアップします。
- Telegram コアにログインしてください: <https://my.telegram.org>。
- **API 開発ツール**にアクセスしてフォームに入力します。
- 基本的なアドレスのほか、ユーザー認証に必要な **api_id** と **api_hash** パラメータも取得できます。
- 現時点では、各番号には 1 つの **api_id** のみを接続できます。

これらの値は Telegram コンポーネントの **Telegram.API** プロパティに設定する必要があります。認証するには、ユーザーまたはボットとして認証できます。Telegram にログインするために設定できる 2 つのプロパティがあります。

- **PhoneNumber:** ユーザーとしてログインする場合は、**電話番号**（国際コード付き）を設定する必要があります。例: +34699123456
- **BotToken:** ボットとしてログインする場合は、Telegram から提供されたトークンをこのプロパティに設定します。
- **DatabaseDirectory:** tdlib データベースの場所を指定できます。空のままにすると、デフォルトの設定が使用されます。

以下のパラメータを設定できます:

- **ApplicationVersion:** アプリケーションバージョン、例: 1.0
- **DeviceModel:** デバイスモデル、例: desktop
- **LanguageCode:** ユーザー言語コード、例: en。
- **SystemVersion:** オペレーティングシステムのバージョン。例: windows。

オプションで、**SetTDJsonPath** メソッドを使用して、tdjson ライブラリが置かれているパスを設定できます。新しい telegram セッションを開始する前にパスを渡すだけです。

Telegram コンポーネントを設定したら、Active プロパティを true に設定するとプログラムは Telegram への接続を試みます。

サンプルコード

```
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;
```

認証

認証コード（SMS ではなく Telegram アプリケーション内で配信）を取得するため、またはパスワードを提供するためにライブラリから呼び出される 2 つのイベントがあります。

OnAuthenticationCode

このイベントは、Telegram が Telegram アプリケーションに認証コードを送信し、ユーザーがそのコードをコピーしてこのイベントの Code 引数に設定する必要があるときに呼び出されます。

```
void OnAuthenticationCode(TObject *Sender, ref string Code)
{
    Code = InputBox("Telegram Code", "Introduce code", "");
}
```

OnAuthenticationPassword

このイベントは、Telegram がユーザーにパスワードの設定を要求するときに呼び出されます。

認可ステータス

認可が開始されると、**OnAuthorizationStatus** イベントで認可のステータスを確認できます。このイベントは認可のステータスに変化があるたびに呼び出されます。ステータスの値の例:

- authorizationStateWaitTdlbParameters
- authorizationStateWaitEncryptionKey
- authorizationStateWaitPhoneNumber
- authorizationStateWaitCode
- authorizationStateLoggingOut
- authorizationStateClosed
- authorizationStateReady

接続ステータス

接続が開始されると、接続のステータスを **OnConnectionStatus** イベントで確認できます。このイベントは接続のステータスに変化するたびに呼び出されます。Status の値の例:

- connectionStateConnecting
- connectionStateUpdating
- connectionStateReady

主要メソッド

TsgcTDLib_Telegram APIコンポーネントはいくつかのTelegramメソッドをサポートしています。以下に最もよく使用されるものを示します。

メソッド	パラメータ	説明
SendMessage	aChatId: メッセージの送信先となる Chat の ID aText: メッセージのテキスト。 InlineKeyboard: オプションのボタン (ボットのみ)。	チャットにテキストメッセージを送信します
SendRichTextMessage	aChatId: メッセージを送信するチャットの ID aText: メッセージのテキスト。 InlineKeyboard: オプションのボタン (ボットのみ)。	チャットにリッチテキストメッセージを送信します。Markdown 構文: • 太字: **bold** • 斜体: <i>__italic__</i> • 取り消し線: ~~--strike--~~ • 下線: <u>~~underline~~</u> • コード: ##code##
SendDocumentMessage	aChatId: メッセージを送信するチャットの ID aFilePath: ドキュメントの完全なファイルパス alInlineKeyboard: オプションのボタン (ボットのみ)。	チャットにドキュメントを送信します。
SendPhotoMessage	aChatId: メッセージを送信するチャットの ID です aFilePath: 写真のフルファイルパスです Width: 写真の幅。 Height: 写真の幅。 InlineKeyboard: オプションのボタン (ボットのみ)。	チャットに写真を送信します。
SendVideoMessage	aChatId: メッセージを送信するチャットの ID aFilePath: 動画の完全ファイルパス aWidth: 動画の幅。 Height: ビデオの幅。 aDuration: 動画の長さ (秒単位)。 alInlineKeyboard: オプションのボタン (ボットのみ)。	チャットにビデオを送信します。
SendInvoiceMessage	aChatId: メッセージを送信するチャットの ID alInvoice: メッセージのテキスト。 alInlineKeyboard: オプションのボタン (ボットのみ)。	請求書を送信します (ボットかつプライベートチャンネルでのみ利用可能)。
EditTextMessage	aChatId: メッセージを送信するチャットの ID aMessageId: 変更するメッセージの ID Text: メッセージのテキスト。 InlineKeyboard: オプションのボタン (ボットのみ)。 ShowKeyboard: オプションボタン (ボットのみ)。	メッセージのテキスト (またはゲームメッセージのテキスト) を編集します
AddChatMember	aChatId: メッセージを送信するチャットの ID。 aUserId: ユーザーの識別子。 aForwardLimit: 新しいメンバーに転送するチャットの以前のメッセージ数。最大 100 件。スーパーグループとチャンネルでは無視されます。	チャットに新しいメンバーを追加します。メンバーはプライベートまたはシークレットチャットには追加できません。チャットの状態がサーバーと同期されるまで、メンバーは追加されません。

AddChatMembers	aChatId: メッセージが送信されるChatのId aUserIds: チャットに追加されるユーザーの識別子。	チャットに複数の新しいメンバーを追加します。現在このオプションはスーパーグループとチャンネルのみ利用できます。このオプションはチャットへの参加には使用できません。チャンネルのメンバーが200人を超えている場合、メンバーを追加することはできません。チャットの状態がサーバーと同期されるまで、メンバーは追加されません。
GetChatMember	aChatId: チャット識別子。 aUserId: ユーザー識別子。	チャットの特定のメンバーに関する情報を返します。
GetBasicGroupFullInfo	aGroupId: 基本グループ識別子	識別子によってベーシックグループの完全な情報を返します。
GetSupergroupMembers	aSuperGroupId: スーパーグループまたはチャンネルの識別子。 aSupergroupMembersFilter: 返すユーザーの種類。デフォルトはnull。 aOffset: スキップするユーザーの数。 aLimit: 返されるユーザーの最大数。最大200人。	スーパーグループまたはチャンネルのメンバーまたは禁止されたユーザーに関する情報を返します。
JoinChatByInviteLink	aLink: インポートする招待リンク；	可能であれば、招待リンクを使用して現在のユーザーをチャットに追加します。チャット状態がサーバーと同期されるまで、新しいメンバーは追加されません。
CreateNewSecretChat	aUserId: ユーザーの識別子。	新しいシークレットチャットを作成します。
CreateNewBasicGroupChat	aUserIds: チャットに追加するユーザーの識別子。 aTitle: 新しい基本グループのタイトル。	新しい基本グループを作成します
CreateNewSupergroupChat	aTitle: 新しいスーパーグループのタイトル alsChannel: True の場合、チャンネルチャットが作成されます。 aDescription: チャットの説明。	新しいスーパーグループまたはチャンネルを作成します。
CreatePrivateChat	aUserId: ユーザーの識別子。 aForce: trueの場合、ネットワークリクエストなしでチャットが作成されます。この場合、チャットのタイプ、タイトル、写真を除くすべての情報が正しくない可能性があります	指定したユーザーに対応する既存のチャットを返します。
GetChats	aOffsetOrder: チャットを返す起点となるチャット順序 aOffsetChatId: チャットを返す起点となるチャット識別子	チャットの順序付きリストを返します。チャットはペア (order, chat_id) で降順にソー

	aLimit: 返されるチャットの最大数です。	トされます（ボットとしてログインされている場合は使用できません）
GetChat	aChatId: チャット識別子	識別子でチャットに関する情報を返します。
GetChatHistory	aChatId: チャット識別子 aFromMessageId: 履歴を取得する開始メッセージの識別子。最後のメッセージから結果を取得するには 0 を使用します。 aOffset: from_message_id からの結果を正確に取得するには 0 を指定し、追加の新しいメッセージを取得するには最大 99 までの負のオフセットを指定します。 aLimit: 返されるメッセージの最大数	チャットのメッセージを返します。メッセージは逆時系列順で返されます
GetUser	aUserId: ユーザー識別子	識別子によってユーザーに関する情報を返します。
AddProxyHTTP	aServer: プロキシのサーバー名。 aPort: プロキシポートの番号。 aUserName: ログイン用のユーザー名。空でもかまいません。 aPassword: ログイン用のパスワード。空でも可。 aHTTPOnly: プロキシが HTTP リクエストのみをサポートし、HTTP CONNECT メソッドによる透過的な TCP 接続をサポートしない場合は True を渡します。	ネットワークリクエスト用の HTTP プロキシサーバーを追加します。認可の前に呼び出すことができます。
AddProxyMTPProto	aServer: プロキシのサーバー名。 aPort: プロキシポートの番号。 aSecret: 16 進数エンコードのプロキシシークレット。	ネットワークリクエスト用 MTPProto プロキシサーバーを追加します。認可前に呼び出し可能です。
AddProxySocks5	aServer: プロキシのサーバー名。 aPort: プロキシポートの番号。 aUserName: ログイン用のユーザー名。空でもかまいません。 aPassword: ログイン用のパスワードです。空でも構いません。	ネットワークリクエスト用の Socks5 プロキシサーバーを追加します。認証前に呼び出すことができます。
EnableProxy	aid: プロキシの ID	プロキシを有効にします。一度に有効にできるプロキシは 1 つだけです。認可前に呼び出し可能です。
DisableProxy		現在有効なプロキシを無効にします。認可の前に呼び出すことができます。
RemoveProxy	aid: プロキシの ID	プロキシサーバーを削除します。認証前に呼び出すことができます。
GetProxies		現在設定されているプロキシのリストを返しま

		す。認証前に呼び出すことができます。
getChatSponsoredMessage	aChatId: チャットのID	チャットに表示されるスポンサーメッセージを返します（チャンネルチャットのみ）。チャットにスポンサーメッセージがない場合は 404 エラーを返します。
ViewMessage	aSponsorChatId: スポンサーチャットの ID。 aMessageId: メッセージの ID	ユーザーがメッセージを閲覧中であることを TDLib に通知します。多くの有用な動作は、メッセージが現在表示されているかどうかによって異なります。
Logout		Telegram からのログアウト。
TDLibSend	aRequest: JSON リクエスト。	JSON プロトコルで任意のリクエストを送信します。

テキストメッセージの送信方法の使用例

```
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");
```

未実装メソッドの送信方法の例

TDLibSendメソッドを使用して任意のJSONメッセージを送信できます。例: Telegramチャットに参加する。

```
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\"}");
```

イベント

OnBeforeReadEvent

このイベントは Telegram API コンポーネントが JSON メッセージを受信し、まだ処理されていない場合に呼び出されます。このイベントを手動で処理する場合、またはコンポーネントによってイベントが処理されないようにする場合は、Handled プロパティを True に設定してください。このイベントはすべてのメッセージをログに記録するためにも使用できます。

OnMessageText

このイベントは新しいテキストメッセージを受信されたときに呼び出されます。MessageText パラメータを読み取ってメッセージテキストのプロパティにアクセスします。

- **ChatId:** チャット識別子。
- **MessageId:** メッセージ識別子。
- **SenderId:** 送信者の識別子。
- **Text:** メッセージのテキスト。

OnMessageDocument

このイベントは、新しいドキュメントメッセージが受信されたときに呼び出されます。MessageDocument にアクセスしてドキュメントプロパティにアクセスします。

- **ChatId:** チャット識別子。
- **MessageId:** メッセージ識別子。
- **SenderId:** 送信者の識別子 (tdlib 1.7.+からSenderChatとSenderUserを読み取ります)。
- **FileName:** ドキュメントの名前。
- **DocumentId:** ドキュメント識別子。
- **LocalPath:** 存在する場合のローカルファイルへの完全パス。
- **MimeType:** ドキュメントの MIME タイプ。
- **Size:** ドキュメントのサイズ。
- **RemoteDocumentId:** リモートドキュメント識別子。

OnMessagePhoto

新しい写真メッセージが受信されたときに呼び出されます。写真のプロパティにアクセスするには MessagePhoto にアクセスします。

- **ChatId:** チャット識別子。
- **MessageId:** メッセージ識別子。
- **SenderId:** 送信者の識別子 (tdlib 1.7.+からSenderChatとSenderUserを読み取ります)。
- **PhotoId:** 写真の識別子。
- **LocalPath:** 存在する場合のローカルファイルへの完全パス。
- **Size:** 写真のサイズ。
- **RemotePhotoId:** リモート写真識別子。

OnVideoPhoto

新しいビデオメッセージを受信したときに呼び出されるイベントです。MessageVideoにアクセスしてビデオプロパティを取得します。

- **ChatId:** チャット識別子。
- **MessageId:** メッセージ識別子。
- **SenderId:** 送信者の識別子 (tdlib 1.7.+からSenderChatとSenderUserを読み取ります)。
- **VideoId:** 写真の識別子です。
- **LocalPath:** 存在する場合のローカルファイルへの完全パス。
- **Width:** ビデオの幅。
- **Height:** 動画の高さ。
- **Duration:** ビデオの再生時間 (秒)。
- **Size:** 動画のサイズ。
- **RemoteVideoId:** リモート写真の識別子。

OnMessageSponsored

このイベントは、新しいSponsoredメッセージが受信されたとき(メソッドgetChatSponsoredMessage)の呼び出し後)に呼び出されます

- **SponsorChatId:** スポンサーチャット識別子。
- **MessageId:** メッセージ識別子。
- **Text:** メッセージのテキスト。

OnNewChat

新しいチャットを受信したときに呼び出されるイベントです。

- **ChatId:** チャット識別子。
- **ChatType:** チャットタイプ (chatTypeSupergroup、chatTypePrivate...)
- **Title:** チャット名。
- **SuperGroupId:** SuperGroupの場合のGroup Id。
- **IsChannel:** チャンネルかどうかを返します。

OnNewCallbackQuery

このイベントは新しい受信コールバッククエリが受信されたときに呼び出されます (ボットのみ)。

- **Id**: 一意のクエリ識別子。
- **SenderId**: クエリを送信したユーザーの識別子。
- **ChatId**: クエリが送信されたチャットの識別子。
- **MessageId**: クエリが発信されたメッセージの識別子。
- **ChatInstance**: メッセージが送信されたチャットに一意に対応する識別子。
- **PayloadData**: 汎用コールバックボタンからのペイロード。
 - **Data**: コールバックボタンにアタッチされたデータ。

OnEvent

このイベントは API コンポーネントが新しいイベントを受信したときに呼び出されます。API コンポーネントで実装されていない一部のイベントを処理するために使用できます。

- **Event**: イベント名 (例: updateOption、updateUser...)
- **Text**: 完全なJSONメッセージ

OnException

Telegram API データの処理中に例外が発生した場合にこのイベントが呼び出されます。

プロパティ

MyId: 現在のユーザーのユーザー識別子を返します。

完全なコードサンプル

以下のコードサンプルを確認してください。Telegram API への接続方法、(Telegram API が必要な場合に) ユーザーにコードを入力させる方法、接続準備ができたならメッセージを送信する方法、受信したテキストメッセージをログに記録する方法を示しています。

```
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 | インラインボタン付き Telegram メッセージの送信

Telegram API では、回答としてオプションを選択するためのインラインボタン付きメッセージを送信できます（このオプションはボットにのみ利用可能です）。

メッセージを送信する前に、クラス `TsgcTelegramReplyMarkupInlineKeyboard` のインスタンスを作成し、作成するボタンごとに `AddButtonTypeCallback` または `AddButtonTypeUrl` メソッドを呼び出します。

使用例

ユーザーがメッセージを好きかどうかを尋ねるメッセージと投票に回答するためのリンクを作成します。`OnNewCallbackQuery` イベントを使用してレスポンスを処理します。

```
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 | ボタン付きボットメッセージの送信

Telegram API を使用すると、ユーザーにデータをリクエストするためのボタン付きメッセージを送信できます (このオプションはボットのみ利用可能です)。

メッセージを送信する前に、**TsgcTelegramReplyMarkupShowKeyboard** クラスのインスタンスを作成し、作成したいボタンごとにメソッド **AddButtonTypeRequestLocation**, **AddButtonTypeRequestPhoneNumber** または **AddButtonTypeText** を呼び出します。

例

PhoneNumber を提供するようにユーザーに求める新しいメッセージを作成します

```
oReplyMarkup = new TsgcTelegramReplyMarkupShowKeyboard();
oReplyMarkup->AddButtonTypeRequestPhoneNumber("Give me your phone");
sgcTelegram->SendTextMessage("123456", "Please provide the information below", null, oReplyMarkup);
oReplyMarkup->Free();
```

Telegram | Telegram 太字メッセージの送信

太字、斜体などのスタイルを使用してテキストメッセージを強調表示できます。スタイル機能付きのテキストメッセージを送信するには **SendRichTextMessage** メソッドを使用してください。このメソッドはテキストメッセージを解析し、API Telegram に必要なエンティティを自動的に追加します。

Markdown Syntax

- 太字 [*]

```
**This is Bold**
```

- 斜体 [_]

```
__This is Italic__
```

- Strike [-]

```
--This is Strike--
```

- 下線 [~]

```
~~This is Underline~~
```

- コード [#]

```
##This is Monospace##
```

Telegram | ボットとしてチャットが見つかりません

ボットとしてログインすると、GetChats メソッドは使用できないため、利用可能なすべてのチャットを取得できません。初めてボットとしてログインして既知のチャットにメッセージを送信しようとする、次のエラーが発生します:

```
{"@type":"error","code":5,"message":"Chat not found"}
```

解決策は、Telegram メッセージを送信する前に **GetChat** メソッドを呼び出し、**ChatId** をパラメータとして渡すことです。チャットデータを取得したら、通常通り Telegram メッセージを送信できます。

注意として、このチャットからボットメッセージを一度も受信したことがない場合、メッセージを送信する前に**最初の一度だけGetChatを呼び出す必要があるのは最初の一回だけです**。アプリケーションを閉じて再起動した場合、Chatはすでにtelegramデータベースに保存されているため、最初にGetChatを呼び出す必要はありません。

Telegram | スポンサーードメッセージ

ユーザーがチャンネルを開くたびに、そのチャンネルで利用可能なスポンサーードメッセージを受け取るために `channels.getSponsoredMessages` を呼び出す必要があります。結果は 5 分間キャッシュする必要があります。

スポンサーメッセージの表示

スポンサーメッセージは、ユーザーが最後のメッセージを過ぎてさらにスクロールした後、チャンネル内の他のすべての投稿の下に表示される必要があります。from_id フィールドで指定されたプロモートされたチャンネルまたはボットは、メッセージの作成者として表示される必要があります。メッセージの下部には以下のボタンのいずれかが含まれている必要があります:

- **View Bot:** ボットが宣伝されている場合。ボタンをタップするとボットとのチャットを開く必要があります。start_param が指定されている場合、アプリはディープリンクメカニズムを使用してボットを開く必要があります。
- **チャンネルを表示:** チャンネルが宣伝されている場合。ボタンをタップするとチャンネルが開きます。
- **View Post:** チャンネルがプロモーションされており、channel_postが指定されている場合。ボタンをタップすると、その特定のチャンネル投稿が開きます。

テキスト全体が画面に表示された後(ボタンを除く)、このsponsoredメッセージのrandom_idで**ViewMessage** メソッドを呼び出す必要があります。

Get Sponsored Messages

スポンサーメッセージが利用可能かどうかをチャンネルに問い合わせるリクエストを送信します。**GetChatSponsoredMessage**メソッドを呼び出すだけです。

```
TsgcTDLlib_Telegram *oTelegram = new TsgcTDLlib_Telegram();
oTelegram->Telegram->API->ApiHash = "ABCDEFGHJKLMN";
oTelegram->Telegram->API->ApiId = "1234";
oTelegram->PhoneNumber = "008745744155";
oTelegram->Active = true;
oTelegram->getChatSponsoredMessage("100");
```

チャットにスポンサーードメッセージがある場合、**OnMessageSponsored** イベントがスポンサーードメッセージの内容とともに呼び出されます。メッセージがない場合は 404 エラーが返されます。

```
private void(TObject *Sender, TsgcTelegramMessageSponsored *MessageSponsored)
{
    DoLog(MessageSponsored->Text);
}
```

スポンサーードメッセージがユーザーに表示された後、**ViewMethod** メソッドを呼び出します。

```
oTelegram->ViewMessage("100", "54653256245");
```

Telegram | Telegram 請求書メッセージを送信する

ボットがインラインモードをサポートしている場合、ユーザーはボットを介して他のチャット（他のユーザーとの1対1のチャットを含む）に請求書を送ることもできます。

請求書メッセージには製品の写真と説明、および目立つ支払いボタンが含まれています。このボタンをタップするとTelegramアプリの特別な支払いインターフェイスが開きます。

ボットは、メソッド**SendInvoiceMessage**を使用して、請求書をメッセージとして送信できます。

```
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 API では SuperGroup のメンバーに関する情報を取得できます。**GetSuperGroupMembers** メソッドを使用して、スーパーグループまたはチャンネルのメンバーや禁止ユーザーに関する情報を取得してください。SupergroupFullInfo.can_get_members が true の場合のみ使用できます。さらに、一部のフィルターには管理者権限が必要な場合があります。

デフォルトでは、このメソッドはグループのすべてのメンバーを返しますが、Filter パラメーターを使用して返されるメンバーをフィルタリングできます。以下のパラメーターがあります：

tsgmFilterNone

デフォルト値。メンバーはフィルタリングされません。

tsgmFilterAdministrators

作成者と 管理者を返します。

tsgmFilterBanned

スーパーグループまたはチャンネルからバンされたユーザーを返します。管理者のみが使用できます。クエリを使用して検索するには引数 aSuperGroupMembersQuery を使用できます。

tsgmFilterBots

スーパーグループまたはチャンネルのボットメンバーを返します。

tsgmFilterContacts

スーパーグループまたはチャンネルのメンバーであるユーザーの連絡先を返します。クエリを使用して検索するには引数 aSuperGroupMembersQuery を使用できます。

tsgmFilterMention

スーパーグループでメンションできるユーザーを返します。

tsgmFilterRecent

最近アクティブなユーザーを逆時系列順に返します。

tsgmFilterRestricted

制限されたスーパーグループメンバーを返します。管理者のみが使用できます。クエリを使用して検索するには引数 aSuperGroupMembersQuery を使用できます。

tsgmFilterSearch

(文字列) クエリによってスーパーグループまたはチャンネルのメンバーを検索するために使用されます。クエリを使用して検索するには引数 aSuperGroupMembersQuery を使用できます。

OnEvent コールバックを使用し、event = "chatMembers" でフィルタリングして結果を読み取ることができます。

```
Telegram->GetSupergroupMembers(1452979380);
```

```
private void OnTelegramEvent(TObject *Sender, const string Event, const string Text)
{
    if (Event == "chatMembers")
    {
        ReadJSON(Text);
    }
}
```


Telegram | Telegram プロキシの追加

Telegram クライアントはプロキシを使用するように設定できます。現在、Telegram は3種類のプロキシをサポートしています。

1. HTTP
2. MTProto
3. Socks5

プロキシの追加

HTTP プロキシを設定するには、まず Telegram の設定にプロキシを追加する必要があります。これを行うには、**AddProxyHTTP** を呼び出し、成功すると追加された新しいプロキシを含むメッセージが返されます。プロキシがリストに追加されたら、**EnableProxy** を呼び出し、確認メッセージで受け取った**プロキシの ID** を渡してください。

```
Telegram->AddProxyHTTP("8.8.8.8", 8080, "", "", true);  
// ... read the confirmation message and save the ID of the proxy.  
Telegram->EnableProxy(2);
```

プロキシを削除

RemoveProxy メソッドを呼び出し、削除したいプロキシの ID を渡します。

Telegram | Telegram ユーザーの登録

Telegram に新しいユーザーを登録するプロセスは非常に簡単です。API ID と API Hash、および新しいアカウントの電話番号が必要です。

Telegramクライアントを設定します:

- API Id
- API Hash
- 新しいTelegramアカウントの電話番号。

クライアントを起動すると新しいコードが電話に送信されます。クライアントは Telegram コードを要求し、正しければ `OnRegisterUser` イベントが呼び出されます。このイベントでユーザーのファーストネームとラストネームを設定すると、登録が完了します。

```
TsgcTDLib_Telegram *oTelegram = new TsgcTDLib_Telegram();
oTelegram->Telegram->API->ApiHash = "ABCDEFGHijklmn";
oTelegram->Telegram->API->ApiId = "1234";
oTelegram->PhoneNumber = "008745744155";
oTelegram->Active = true;

void OnTelegramAuthenticationCode(TObject *Sender, ref string Code)
{
    Code = "code sent to phone";
}

void OnTelegramRegisterUser(TObject *Sender, ref string FirstName, ref string LastName)
{
    FirstName = "first name";
    LastName = "last name";
}
```

RCON

RCON

Source RCON プロトコルは Source Dedicated Server が使用する TCP/IP ベースの通信プロトコルで、「リモートコンソール」 (RCON) 経由でサーバーにコンソールコマンドを発行できます。RCON の最も一般的な使用方は、サーバーが動作しているマシンに直接アクセスせずに、サーバーオーナーがゲームサーバーを制御できるようにすることです。

設定

RCON_Options では次のプロパティを設定できます:

- **Host:** サーバーのリモートアドレス。
- **Port:** サーバーのリスニングポート。
- **Password:** サーバーに対して認証するために使用するシークレット文字列。

Connect

サーバーへの接続/切断には **Active** プロパティを使用します。

ActiveがTrueに設定されると、クライアントはサーバーへの接続を試みます。接続できれば、提供されたパスワードを使用して認証を試みます。

サーバーは認証要求に対する応答を送信します。 **OnAuthenticate** イベントが呼び出され、Authenticateパラメータを使用して認証が成功したかどうかを読み取れます。

コマンドを送信する

ExecCommand メソッドを使用してサーバーにコマンドを送信します。レスポンスは **OnResponse** イベントで利用できます。

```
TsgcLib_RCON oRCON = new TsgcLib_RCON();
oRCON->RCON_Options->Host = "127.0.0.1";
oRCON->RCON_Options->Port = 25575;
oRCON->RCON_Options->Password = "test";
oRCON->Active = true;

void OnAuthenticate(TObject *Sender, bool Authenticated, const TsgcRCON_Packet *aPacket)
{
    if (Authenticated == true)
    {
        DoLog("#authenticated");
    }
    else
    {
        DoLog("#not authenticated");
    }
}

void OnResponse(Object *Sender, const string aResponse, const TsgcRCON_Packet *aPacket)
{
    DoLog(aResponse);
}
```

CryptoHopper

CryptoHopper

CryptoHopper は、Bitcoin、Ethereum、Litecoin などの取引とポートフォリオ管理を自動化できる自動暗号通貨取引ボットです。

設定

デベロッパーアカウントが必要で、承認されると新しいアプリの作成を開始できます。API は OAuth2 を使用して認証するため、アプリから **client_id** と **client_secret** を取得できます。

```
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";
```

メソッド

CryptoHopper は HTTPS プロトコルを使用して API にリクエストを送信します。一部のメソッドは認証が必要（注文の発注、ユーザーデータの取得など）で、他のメソッドは公開されています（例：取引所データの取得）。

関数は CryptoHopper のレスポンスを返し、エラーがある場合は例外が発生します。

Hoppers

基本的なホッパー操作を管理します。

メソッド	Arguments	説明
GetHoppers		ユーザーの Hopper を取得します。
CreateHopper	aBody: 設定 JSON テキスト。	新しいホッパーを作成します。
GetHopper	aid: hopper id	Retrieve Hopper
DeleteHopper	aid: hopper id	Delete Hopper
UpdateHopper	aid: ホッパー ID aBody: 設定 JSON テキスト。	Hopper を更新します。

注文

Hopperの注文を管理します。

メソッド	Arguments	説明
GetOpenOrders	ald: hopper id	ホッパーのすべてのオープンオーダーを取得します。
CreateNewOrder	ald: ホッパーID aOrder: TsgcHTTPCTHOrder のインスタンス	新しい買いまたは売り注文を作成します。売りの場合は、sell エンドポイントを使用してください。
PlaceMarketOrder	ald: ホッパー ID aOrderSide: cthosBuy または cthosSell。 aCoin: コイン名 (例: EOS) aAmount: 注文サイズ。	成行注文を発注します。
PlaceLimitOrder	ald: ホッパー ID aOrderSide: cthosBuy または cthosSell。 aCoin: コイン名 (例: EOS) aAmount: 注文サイズ。 aPrice: 指値価格。	Limit Orderを発注します
DeleteOrder	ald: ホッパー ID aOrderId: 注文 ID	選択したhopperの注文を削除します。
DeleteAllOrders	ald: hopper id	選択したホッパーのすべてのオープン注文を削除します。
GetOpenOrder	ald: ホッパー ID aOrderId: 注文 ID	idによってhopper内のオープン注文を取得します。
CancelOrder	ald: ホッパー ID aOrderId: 注文 ID	オープン注文をキャンセルします。

ポジション

Hopper のポジションを管理します。

メソッド	Arguments	説明
GetPosition	ald: hopper id	ホッパーのオープンポジションを取得します。

取引

ホッパーのトレード履歴。

メソッド	Arguments	説明
GetTradeHistory		ホッパーの取引履歴を取得します。
GetTradeHistoryById	aId: ホッパーID aTradeId: 取引ID	ホッパーの ID でトレードを取得します。

エクスチェンジ

CryptoHopper で利用可能な取引所の情報を取得する

メソッド	Arguments	説明
GetExchange		Cryptohopper で利用可能なすべての取引所を取得します。
GetAllTickers	aExchange: エクスチェンジ名	すべてのペアのティッカーを取得します。
GetMarketTicker	aExchange: エクスチェンジ名 aPair: ペア名	マーケットペアの Ticker を取得します。
GetOrderBook	aExchange: エクスチェンジ名 aPair: ペア名 aDepth: 注文板の深さ	選択した取引所、市場、オーダーブック深度のオーダーブックを取得します。

Webhooks

ホッパーのトレード履歴。

メソッド	Arguments	説明
CreateWebhook	aURL: webhook URL aMessageTypes: カンマで区切られたメッセージタイプ。	Webhook を更新または作成します。
DeleteWebhook	aURL: webhook の URL	既存のWebhookを削除します。

Signals

CryptoHopper APIにシグナルを送信します。

メソッド	Arguments	説明
SendSignal	aSignal: シグナルを送信するために必要なすべてのフィールドを含むクラスです。	シグナルを送信します
SendTestSignal	aSignal: シグナルを送信するために必要なすべてのフィールドを含むクラスです。	テスト信号を送信します
GetSignalStats	aSignalId: シグナルの ID。 aExchange: オプション。エクスチェンジ名。	シグナル統計の一部を取得します。

How to Update Cryptohopper Config

UpdateHopper メソッドを使用してホッパー設定を更新します。このメソッドはオーバーロードされており、JSON 文字列を渡すか、TsgcHTTPCTHopper オブジェクトを使用してプロパティでホッパープロパティを有効または無効にすることができます。

```
public string EnableHopper()
{
    TsgcHTTPCTHopper *oHopper = new TsgcHTTPCTHopper();
    try
    {
        oHopper->Enabled = 1;
        result = Cryptohopper->UpdateHopper("1234", oHopper);
    }
    finally
    {
        oHopper->Free();
    }
}
```

Webhook の設定方法

Webhook を使用すると、ホッパーで何かが起きたときに通知を受け取ることができます。Webhook には、cryptohopper から送信されたすべてのメッセージを URL アドレスでリッスンするパブリック HTTPS サーバーが必要です。パブリックサーバーは SSL 証明書で保護されている必要があります（自己署名証明書は許可されていません）。

最初に Webhook を作成する必要があります。Cryptohopper クライアントの Webhook プロパティを設定し、サーバーがリッスンするホストとポートを指定してください。次に SSLOptions プロパティで証明書を設定します。

例: パブリックIPアドレスは1.1.1.1、リッスンポートは443になります。証明書は、ファイル名sgc.pemのPEMファイルとして、パスワードなしで保存されます。

```
/* OAuth2 */
cryptohopper->CryptohopperOptions->OAuth2->ClientId = "client_id";
cryptohopper->CryptohopperOptions->OAuth2->ClientSecret = "client_secret";
cryptohopper->CryptohopperOptions->OAuth2->LocalIP = "127.0.0.1";
cryptohopper->CryptohopperOptions->OAuth2->LocalPort = 8080;
/* Webhook */
cryptohopper->CryptohopperOptions->Webhook->Enabled = True;
cryptohopper->CryptohopperOptions->Webhook->Host = "1.1.1.1";
cryptohopper->CryptohopperOptions->Webhook->Port = 443;
cryptohopper->CryptohopperOptions->Webhook->ValidationCode = "1234";
cryptohopper->CryptohopperOptions->Webhook->SSLOptions->CertFile = "sgc->pem";
cryptohopper->CryptohopperOptions->Webhook->SSLOptions->KeyFile = "sgc->pem";
cryptohopper->CryptohopperOptions->Webhook->SSLOptions->RootCertFile = "sgc->pem";
cryptohopper->CryptohopperOptions->Webhook->SSLOptions->Password = "";
cryptohopper->StartWebhook();
```


RTCMultiConnection

RTCMultiConnection

RTCMultiConnection は、ピアツーピアアプリケーション（画面共有、音声/ビデオ会議、ファイル共有、メディアストリーミングなど）向けの WebRTC JavaScript ライブラリです。

設定

RTCMultiConnection はシグナリング用の WebSocket サーバーを必要とするため、RTCMultiConnection のサーバープロパティを WebSocket サーバー ([TsgcWebSocketHTTPServer](#) など) にリンクしてください。設定が必要なプロパティを以下に示します。

サーバー

Host: WebSocket サーバーのパブリック IP アドレスまたは DNS 名です。

Port: WebSocketサーバーの リッスンポートです。

IceServers

ピア間の通信を可能にするICEサーバー（STUN/TURN）の設定です。例：

```
[
  {
    "urls": "stun:www.yourstun.com"},
  {
    "urls": "turn:www.yourturn.com",
    "username": "user",
    "credential": "secret"
  }
]
```

VideoResolution

ここでは、ビデオ会議のビデオ解像度を設定できます。解像度が高いほど、接続に必要な帯域幅が増加します。

HTMLDocuments

各アプリケーションに対して、このコンテンツを提供する HTML ページの名前を設定します。

例: サーバーがWebサイト www.webrtc.com のポート 8443 で動作しており、HTMLDocuments.VideoConferencing = /RTCMultiConnection-VideoConferencing.html の場合、VideoConferencingにアクセスするURLは次のようになります。

<https://www.webrtc.com:8443/RTCMultiConnection-VideoConferencing.html>

WebRTC にはセキュアな接続（HTTPS）が必要です。証明書の使用が求められます。詳細は[サーバー SSL](#)をお読みください。

アプリケーション

名前	説明
ビデオ会議	メッシュネットワーキングモデルを使用したマルチユーザー（多対多）ビデオチャット。
ScreenSharing	スタートポロジを使用したマルチユーザー（一対多）画面共有。

VideoBroadcasting

スタートポロジーを使用した、マルチユーザー（1対多）のビデオブロードキャストです。

WebPush

[RFC 8030](#)
[RFC 8291](#)

WebPush プロトコルは **RFC 8030** (HTTP Push を使用した配信) および **RFC 8291** (メッセージ暗号化) によって定義されています。

Web Push は、**ウェブブラウザへのプッシュ通知配信のための標準化されたプロトコル**です。Push API を使用しており、これはウェブサイトがプッシュメッセージを登録および受信できる標準のウェブ API です。Push API により、ウェブサイトはユーザーが積極的にサイトを閲覧していない場合でも、ユーザーのブラウザにプッシュメッセージを送信できます。

Web Pushを使用するには、Webサイトはまず**ユーザーのブラウザからpushサブスクリプションを取得**する必要があります。サブスクリプションは、一意のエンドポイントURLと暗号化キーで構成されます。エンドポイントURLは、Webサイトがユーザーのブラウザにpushメッセージを送信するために使用できるURLであり、暗号化キーはpushメッセージの暗号化と復号化に使用されます。

ウェブサイトが**プッシュサブスクリプションを取得**すると、エンドポイント URL への HTTP リクエストを行うことでユーザーのブラウザに**プッシュメッセージを送信**できます。プッシュメッセージは Web Push Protocol Message と呼ばれる特別な形式で送信されます。これは一連のヘッダーとペイロードから構成されています。ヘッダーには暗号化キーやメッセージの TTL (有効期限) などの情報が含まれ、ペイロードにはメッセージの実際のコンテンツが含まれます。

ユーザーのブラウザがプッシュメッセージを受信すると、まず暗号化キーを使用して**メッセージを復号化**します。その後、通知を解除するか Web サイトを開くなどのユーザーが実行できる追加のアクションとともに**ユーザーに通知を表示**します。

プッシュメッセージのセキュリティとプライバシーを確保するために、Web Push はエンドツーエンド暗号化を使用し、セキュアな接続 (例: HTTPS) を通じてプッシュサブスクリプションを取得する必要があります。さらに、このプロトコルはプッシュメッセージの送信者を認証し、悪用を防ぐメカニズムを提供します (例: ウェブサイトがユーザーに送信できるプッシュメッセージの数を制限するなど)。

コンポーネント

WebPush をサポートする2つのコンポーネントがあります：

- **TsgcWSAPIServer_WebPush:** サーバー側で WebPush プロトコルを実装し、ユーザーへの許可のリクエスト、サブスクリプションの登録、通知の送信などを行えます。このコンポーネントにはすでに通知を送信するための WebPush クライアントが内包されています。
- **TsgcWebPush_Client:** クライアント側で WebPush プロトコルを実装し、デスクトップおよびモバイル Web 経由でユーザーに通知を送信できます。キーとエンドポイントがすでにあり、サブスクライブされたクライアントに WebPush メッセージを発行するだけの場合に便利です。

TsgcWSAPIServer_WebPush

TsgcWSAPIServer_WebAuthn コンポーネント。

はじめに

TsgcWSServer_API_WebPush は WebPush サブスクリプションを処理する機能を提供するコンポーネントです。WebPush はブラウザで実行される Web アプリケーションにリアルタイム通知を配信するためのプロトコルです。このコンポーネントはサブスクリプションを管理し、サブスクライブしたクライアントに通知を送信するために使用できます。TsgcWSServer_API_WebPush クラスが提供するプロパティ、イベント、メソッドとコード例を以下に示します。

設定

1. **TsgcWSServer_API_WebPush** を Server プロパティを使用してWebSocketサーバーにアタッチします。
2. **WebPush.VAPID** プロパティで**公開鍵**と**秘密鍵**を設定します。（登録ユーザーは Windows 用の VAPID キーを生成する実行ファイルをダウンロードできます）。
3. **OpenSSL 3.0.0** バージョンのデプロイが必要です。
4. **WebPush.Endpoints** プロパティで WebPush サブスクリプションを処理するための独自のエンドポイントを定義できます。デフォルトでは、"/sgcWebPush.html" エンドポイントにアクセスすると、WebPush 通知のサブスクリプションができるシンプルなウェブページが表示されます。
5. サーバーを起動し、テストするために設定されたエンドポイントにアクセスします。

リファレンス

- プロパティ — コンポーネントで利用可能な3つのプロパティ。
- メソッド — 3つのパブリックメソッド。
- イベント — 処理可能な3つのイベント。

TsgcWebPush_Client

TsgcWebPush_Client は、サブスクリプションの詳細を取得した後に通知を送信できるクラスです。

はじめに

TsgcWebPush_Client は、サブスクリプションの詳細を取得した後に通知を送信できるクラスです。

以下に、WebPush サブスクリプションのエンドポイント、公開鍵、認証シークレットを指定して通知を送信する WebPush クライアントの使用例を示します。

```
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();
    }
}
```

リファレンス

- プロパティ — コンポーネントで利用可能な3つのプロパティ。
- メソッド — 1つのパブリックメソッド。
- イベント — 処理可能なイベントが 0 個。

Extensions

WebSocket プロトコルは拡張可能なように設計されています。WebSocket クライアントは拡張機能をリクエストでき、WebSocket サーバーはクライアントがリクエストした一部またはすべての拡張機能を受け入れることができます。

サポートされている拡張機能:

1. [Deflate-Frame](#): WebSocket フレームを圧縮します。
2. [PerMessage-Deflate](#): WebSocket メッセージを圧縮します。

拡張機能 | PerMessage-Deflate

PerMessage は WebSocket プロトコル拡張です。この拡張がサーバーとクライアントの両方でサポートされている場合、両者は送信メッセージを圧縮できます：

- 圧縮メソッドとして Deflate を使用します。
- 圧縮はアプリケーションデータにのみ適用されます (制御フレームとヘッダーには影響しません)。
- サーバーとクライアントは、どのメッセージを圧縮するかを選択できます。

最大ウィンドウビット

この拡張機能により、LZ77 アルゴリズムが使用するスライディングウィンドウのサーバー側とクライアント側のサイズをカスタマイズできます (8 から 15 の間)。この値が大きいほど重複を検出して除去する可能性が高まりますが、メモリと CPU サイクルをより多く消費します。デフォルト値は 15 です。

No Context Take Over

デフォルトでは、圧縮と解凍に以前のメッセージが使用されます。メッセージが類似している場合、これにより圧縮率が向上します。有効にすると、各メッセージは自身のメッセージデータのみを使用して圧縮されます。デフォルトでは無効です。

MemLevel

この値は、サーバーとクライアント間でネゴシエートされません。1に設定すると、最小限のメモリを使用しますが、圧縮アルゴリズムが遅くなり、圧縮率が低下します。9に設定すると、最も多くのメモリを使用し、最高のパフォーマンスを提供します。デフォルトでは1に設定されています。

* Rad Studio XE2に付属するIndy バージョンは、初期化関数とのzlib バージョンの不一致により例外を発生させます。これを修正するには、Indyバージョンを最新のものに更新してください。

拡張機能 | Deflate-Frame

これは WebSocket プロトコルの拡張機能であり、WebSocket プロトコルを使用して送信されるフレームの圧縮を可能にします。Chrome や Safari などの WebKit ブラウザでサポートされています。この拡張機能はサーバーとクライアントコンポーネントの両方でサポートされています。

この拡張機能は非推奨になりました。

* Rad Studio XE2に付属するIndy バージョンは、初期化関数とのzlib バージョンの不一致により例外を発生させます。これを修正するには、Indyバージョンを最新のものに更新してください。

MCP

MCP (Model Context Protocol) は、言語モデルと外部システム間の**コンテキスト情報の交換**を定義および管理するために設計された標準化されたプロトコルです。

AIモデルがネイティブ環境を超えた外部ツールやデータソースを**理解**し、**拡張**し、**連携**できるような、構造化された安全で動的なコミュニケーションを可能にします。

MCP はコンテキストメタデータを渡すための**統一インターフェース**を提供し、モデルが *WebAuthn* が認証フローを標準化するのと同様に、予測可能で相互運用可能な方法で指示を解釈し、データを取得し、タスクを実行できるようにします。

目的

MCP は、モデル、クライアント、ツール間の**コンテキストの断片化と統合の不一致**の課題を解決するために開発されました。

MCP では、準拠したクライアントが共有の明確に定義された構造を通じて、MCP 対応のモデルまたはサービスと通信できます。

一般的なユースケースには以下が含まれます:

- モデルクエリへの**コンテキストメタデータ**（ユーザーセッション、アプリの状態、パーミッションなど）のアタッチ。
- **関数またはツール定義**を動的に交換します。
- モデルセッション間での**ステートフルなインタラクション**の管理。
- LLM、SDK、バックエンドサービス間の**安全な相互運用性**の確立。

コンポーネント

- **TsgcWSAPIServer_MCP**: このコンポーネントは、ツール、プロンプト、リソースのリクエスト、認証、イベントを使用したフロー制御など、MCP サーバーを実装するためのシンプルかつ強力なソリューションを提供します。
- **TsgcWSAPIClient_MCP**: このコンポーネントは、ツール、プロンプト、リソースリクエストなどの機能を備えた MCP クライアント実装のためのシンプルかつ強力なソリューションを提供します。

TsgcWSAPIServer_MCP

TsgcWSAPIServer_MCP コンポーネント。

はじめに

TsgcWSAPIServer_MCP は、sgcWebSockets HTTP サーバー エンドポイントを通じて Model Context Protocol (MCP) を公開します。このコンポーネントは、着信 HTTP リクエストを TsgcAI_MCP_Server エンジンとブリッジするため、MCP 互換のクライアントがセッションをネゴシエートし、prompts/resources/tools を列挙し、JSON-RPC スタイルの呼び出しを通じてツールを呼び出すことができます。

設定

- **コンポーネントの配置:** TsgcWSAPIServer_MCP コンポーネントは、Server プロパティを使用して、HTTP サーバー、 [TsgcWebsocketHTTP_Server](#) または [TsgcWebsocketServer_HTTPAPI](#) にアタッチする必要があります。
- **エンドポイントの選択:** デフォルトの /mcp パスを変更する必要がある場合は EndpointOptions.Endpoint を調整します。ドキュメントがこのパスと一致するリクエストは MCP 呼び出しとして扱われます。それ以外は親 API のリソース処理にフォールバックします。
- **MCP 設定の構成:** MCPOptions.ServerInfo を使用してサーバーの名前/バージョンを設定し、MCPOptions.SessionTimeout でセッションの有効期間を制御し、MCPOptions.AuthenticationOptions で JSON-RPC 呼び出しが処理される前に API キーまたはカスタム HTTP ヘッダーを要求します。
- **プロンプト、ツール、リソース、リソーステンプレートの登録:** コンポーネントが公開する読み取り専用の Tools、Prompts、Resources、ResourceTemplates リストを設定します。
- **イベントの処理:** 以下に説明する公開イベントを接続して、セッションライフサイクル、プロンプト/リソース解決、またはツール実行のためのビジネスロジックをプラグインします。例外をインターセプトして HTTP レスポンスコードをカスタマイズできます。

組み込み HTTP エンドポイント（設定不要）

コンポーネントが HTTP サーバーにアタッチされている場合、有効にするための新しいプロパティなしで、次のエンドポイントと動作が /mcp とともに自動的に提供されます。これにより、CORS 上で Dynamic Client Registration を伴う OAuth 2.1 を必要とする **claude.ai** などの Web ベースの MCP コネクタとの、すぐに使える互換性が実現します。

- **オリジンリフレクションによる CORS サポート。** 処理されたパスへの OPTIONS プリフライトは、次のヘッダーを含む 204 No Content を返します:
 - Access-Control-Allow-Origin: リクエストの Origin ヘッダーのエコー、または存在しない場合は *。
 - 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
 すべての非OPTIONS レスポンスには Access-Control-Allow-Origin、Access-Control-Expose-Headers、および Strict-Transport-Security も含まれます。
- **401 Unauthorized と www-Authenticate。** MCPOptions.AuthenticationOptions.ApiKey（または CustomHeader）が有効で、リクエストに認証情報がないか誤っている場合、サーバーは 401 と次の形式のヘッダーで応答します:


```
Bearer realm="MCP", resource_metadata="https://<host>/.well-known/oauth-protected-resource"
```
- **OAuth 2.1 ディスカバリーおよびフローエンドポイント。** 以下のエンドポイントは /mcp と同じ HTTP サーバーで提供され、PKCE（S256）、認証コードおよびリフレッシュトークングラント、動的クライアント登録をサポートします:

パス	仕様	動作
<code>GET /.well-known/oauth-authorization-server</code>	RFC 8414	JSON 認可サーバーのメタデータ。PKCE (S256)、 <code>authorization_code</code> + <code>refresh_token</code> グラント、パブリッククライアント + <code>client_secret_basic</code> 認証メソッド、 <code>/oauth/register</code> での動的登録、 <code>mcp</code> スcopeをアドバタイズします。
<code>GET /.well-known/oauth-protected-resource</code>	RFC 9728	MCP エンドポイントを保護リソースとして記述し、このサーバーを認可サーバーとして指定する JSON。
<code>POST /oauth/register</code>	RFC 7591 (DCR)	JSON クライアント登録ボディ (<code>client_name</code> 、 <code>redirect_uris</code>) を受け入れます。生成された <code>client_id</code> を発行します (<code>client_secret</code> なし — パブリッククライアント)。登録済みクライアント情報レスポンスとともに <code>201</code> を返します。
<code>GET /oauth/authorize</code>	RFC 6749 + RFC 7636	HTML 同意ページをレンダリングします。必須クエリパラメーター: <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	同意フォームを処理します。ユーザーは承認するために <code>MCPOptions.AuthenticationOptions.ApiKey.Value</code> と同じ値を入力する必要があります。承認時は <code>redirect_uri</code> に <code>?code=...&state=...</code> でリダイレクトします。拒否または間違いのキーの場合、 <code>?error=access_denied&state=...</code> でリダイレクトするか、フォームを再レンダリングします。
<code>POST /oauth/token</code>	RFC 6749 + RFC 7636	フォームエンコードされたボディ。 <code>grant_type=authorization_code</code> (PKCE を検証: <code>BASE64URL(SHA256(code_verifier)) == code_challenge</code>) と <code>grant_type=refresh_token</code> をサポートします。1 時間の Bearer アクセストークンとリフレッシュトークンを発行します。認可コードは 60 秒後に期限切れになり、1 回使い捨てです。

- **Bearer トークン検証の拡張。** `MCPOptions.AuthenticationOptions.ApiKey.Enabled` が `True` の場合、`/mcp` リクエストの `Authorization: Bearer <x>` ヘッダーは、`<x>` が構成された静的な `ApiKey.Value`、または現在有効な OAuth 発行のアクセストークン (上記のフローで発行) のいずれかに一致する場合に受け入れられます。
- **インメモリストレージ。** OAuth クライアント、コード、トークンはメモリにのみ保持されます。サーバーの再起動時に失われます。
- **`onMCPHttpRequest` によるオーバーライド。** `onMCPHttpRequest` に接続されたハンドラは、`Handled := True` を設定することで、上記のいずれかの動作をショートサーキットできます。

リファレンス

- プロパティ — コンポーネントで利用可能な 5 つのプロパティ。
- メソッド — 9 つのパブリックメソッド。
- イベント — 処理可能な 12 個のイベント。

MCP サーバー | セッション

サーバーとクライアント間の正常な初期化後、新しいセッションが作成されます。セッションはリクエスト/レスポンスフローでどのセッションが使用されているかを識別するために、クライアントとサーバー間のリクエストおよびレスポンスで常々送信されます。HTTP トランスポートを使用する場合、セッションのライフタイムは単一の HTTP リクエストとレスポンスです。つまり、レスポンスが送信されるとセッションは閉じられます。

セッションオブジェクトはすべてのイベントで渡されます。セッションが作成または削除されたことを知るには、以下のイベントを使用してください：

- **OnMCPSessionNew**: 新しいセッション id が作成されるたびにこのイベントが呼び出されます。
- **OnMCPSessionEnd**: セッションが終了したときにこのイベントが呼び出されます。

OnMCPSessionNew

このイベントは、新しいセッションが作成されたときに呼び出されます。Session パラメータにアクセスして、セッションの ID、セッションの有効期限などを取得できます。

OnMCPSessionEnd

セッションが終了したときにイベントが呼び出されます。

```
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 サーバー | ツール

TsgcWSAPIServer_MCP は sgcWebSockets 上で Model Context Protocol (MCP) サーバー機能を公開する高レベルコンポーネントです。**MCP クライアントが MCP Server Tools 仕様に従ってツールをリクエストすると**、このコンポーネントが HTTP ゲートウェイ、セッションライフサイクル、JSON-RPC シリアライゼーションを調整します。これにより Delphi コードはビジネスロジックの実装に集中できます。

EndpointOptions.Endpoint で定義された MCP エンドポイントにヒットする受信 HTTP リクエストは **TsgcWSServer_API_MCP** によってインターセプトされます。コンポーネントは MCP セッションを作成（または再開）し、リクエストを基礎となる TsgcAI_MCP_Server エンジンに転送し、最終的に仕様で要求される mcp-session-id ヘッダーを保持しながら JSON-RPC レスポンスを HTTP クライアントに書き戻します。

ツールリスト

サーバーは **TsgcAI_MCP_ToolsList** にツールのインメモリカタログを保持しています。ツールは名前によって一意性が保証され、説明を公開し、仕様の tools.list メソッドへのレスポンスで発行される JSON スキーマに似た入力説明を提供します。クライアントが tools.list を呼び出すと、TsgcWSAPIServer_MCP はリクエストを読み込み、現在のカタログをシリアル化して HTTP ステータスコード 200 で返します。

シンプルな計算機ツールを公開するコード例:

```
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");
}
```

クライアントに送信された準拠した tools.list レスポンスは MCP JSON スキーマペイロードを反映します。例:

```
{
  "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" }
          }
        }
      }
    ]
  }
}
```

ツールリクエスト

クライアントが tools.call JSON-RPC リクエストを発行すると、**TsgcWSAPIServer_MCP** は、**OnMCPRequestTool** イベントを発生させる前に、強く型付けされたリクエストオブジェクト(ツール名と提供された引数を含む)をハイドレートします。ハンドラはレスポンスペイロードを設定し、それが成功HTTPステータスコードとともにクライアントにシリアライズされて返されます。

典型的なハンドラーは次のようになります:

```
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;
    }
}
```

生成される JSON-RPC レスポンスは、仕様の tools.call スキーマに従い、1 つ以上のコンテンツパーツとオプションの構造化ペイロードをラップします。コンポーネントがレスポンスをクリアして HTTP コードを設定するため、追加の配管は不要です。

パブリックAPIサーフェス

いくつかのヘルパークラスが、ツールカタログとイベントレスポンスのヘルパーを公開します。次のリストは、本番コードから通常呼び出されるメソッドをまとめたものです。

TsgcAI_MCP_ToolsList

- **AddTool(const Name, Description):** ツールエントリを作成または置き換え、基盤となる **TsgcAI_MCP_Tool** を返してスキーマとデフォルトを設定できるようにします。
- **Clear:** 登録されているすべてのツールを削除します。ディスクから設定を読み込む際にカタログを再構築する前に呼び出してください。
- **Count / Items[Index]:** 列挙、診断、またはカスタムシリアルライズのために、リストの内容を公開します。

TsgcAI_MCP_Tool

- **InputSchema.Properties.AddProperty(Name, Required, JsonType, Description):** **tools.list**を通じてアドバタイズされるJSON入力ペイロードを記述します。
- **InputSchema.Properties.Clear:** アプリケーションが実行時にツール機能を更新するときにスキーマをその場で再構築します。
- **説明:** クライアントと共有する人間が読めるサマリーを格納する書き込み可能なプロパティ。

TsgcAI_MCP_Response_ToolsCall

- **Result.Content.AddText(Value):** ツールレスポンスにテキストブロックを追加します。複数回呼び出すとマルチパートレスポンスが生成されます。
- **Result.Content.AddImage(Data, MimeType):** ツールレスポンスに画像ブロックを追加します。複数回の呼び出しはマルチパートのレスポンスを生成します。
- **Result.Content.AddAudio(Data, MimeType):** ツールレスポンスに音声ブロックを追加します。複数回の呼び出しでマルチパートのレスポンスが生成されます。
- **Result.Content.AddEmbeddedResource(Uri, Title, Text, MimeType):** ツールレスポンスに埋め込みリソースブロックを追加します。複数回呼び出すとマルチパートレスポンスが生成されます。
- **Result.Content.AddResourceLink(Uri, Name, MimeType):** ツールレスポンスにリソースリンクブロックを追加します。複数回呼び出すとマルチパートレスポンスが生成されます。

- **Result.Content.Clear**：新しいコンテンツを追加する前に以前に生成された出力をリセットします。
- **Result.IsError**：ツール呼び出しを実行できない場合はこのフラグを **True** に設定してください。フレームワークは MCP エラーオブジェクトをシリアル化します。

高度な使用例

次のスニペットは、入力を検証し、複数のテキストブロックを返し、パラメータが無効な場合に **IsError** を反転させるジオルックアップツールを構築します。

```
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

MCP では、プロンプトはサーバーが公開する再利用可能なテンプレートまたはワークフローです。ユーザーが制御します（クライアントがどのプロンプトを呼び出すかを選択します）。主なメソッドは2つです。

- **prompts/list:** サーバーが公開するすべての利用可能なプロンプト定義のリストを要求します。TsgcWSAPIServer_MCPコンポーネントによって内部的に処理されます。
- **prompts/get:** 特定のプロンプトのコンテンツとオプションのメタデータ（通常はクライアントの対話方法を説明するメッセージシーケンス）を取得します。サーバーが新しいリクエストを受信したときに **OnMCPRequestPrompt** イベントが呼び出されます。

さらに、サーバーが宣言した機能で `prompts.listChanged = true` を設定している場合、プロンプトカタログが更新されると `notifications/prompts/list_changed` のような通知を送信する場合があります。

クライアントが `prompts/get` をリクエストすると、引数（あれば）を送信し、コンテンツブロック（通常はテキストですが、リソースや画像を埋め込む場合もあります）を含む一連のメッセージ（ユーザーまたはアシスタントのロール）を期待します。

問題がある場合（プロンプトが見つからない、引数が無効など）、サーバーは -32602（無効なパラメーター）や -32603（内部エラー）などの JSON-RPC エラーコードを返します。

プロンプトリスト

サーバーは **TsgcAI_MCP_PromptsList** にプロンプトのインメモリカタログを保持します。プロンプトは名前であり、意であることが保証され、説明を公開し、仕様の `prompts.list` メソッドへのレスポンスで送られる JSON スキーマライクな入力説明を提供します。クライアントが `prompts.list` を呼び出すと、TsgcWSAPIServer_MCP はリクエストをロードし、現在のカタログをシリアル化して 200 HTTP ステータスコードで返します。

コードレビュープロンプトをパブリッシュするコード例：

```
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
    );
}
```

プロンプトリクエスト

クライアントが `prompts.get` JSON-RPC リクエストを発行すると、**TsgcWSAPIServer_MCP** は強く型付けされたリクエストオブジェクト（プロンプト名と提供された引数を含む）をハイドレートしてから、**OnMCPRequestPrompt** イベントを発生させます。ハンドラーはレスポンスペイロードを設定し、成功の HTTP ステータスコードとともにクライアントにシリアル化して返されます。

典型的なハンドラーは次のようになります：


```
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');");
    }
}
```

パブリックAPIサーフェス

プロンプトはいくつかのヘルパークラスを通じてモデル化されます。以下のメソッドは、プロンプトテンプレートを管理し、**prompts/get**レスポンスを構成する際に最もよく使用されます。

TsgcAI_MCP_PromptsList

- **AddPrompt(const Name, Description):** 新しいプロンプトエントリを登録し、引数とメッセージを設定できるように **TsgcAI_MCP_Prompt** オブジェクトを返します。
- **Clear:** カタログを消去します。プロンプトが構成ファイルから読み込まれる起動時に便利です。
- **Count / Items[Index]:** 診断、UI表示、または手動でのシリアルライズのために、リストの内容へのアクセスを提供します。

TsgcAI_MCP_Prompt

- **Arguments.AddArgument(Name, Description, Required):** プロンプトが受け付ける構造化入力を宣言します。
- **Arguments.Clear :** ランタイムでプロンプトが変わる場合に引数リストを再構築します。
- **Messages.AddText(Role, Text):** プロンプト定義の一部としてクライアントに送り返される会話ターンを事前に設定します。

TsgcAI_MCP_Response_PromptsGet

- **Result.Description:** プロンプト ペイロードに付随する短い人間が読めるサマリー。
- **Result.Messages.AddText(Role, Text):** プロンプトが取得されるときに追加のメッセージを動的に挿入します。
- **Result.Messages.AddImage(Data, MimeType):** プロンプトレスポンスに画像ブロックを追加します。複数回呼び出すとマルチパートレスポンスが生成されます
- **Result.Messages.AddAudio(Data, MimeType):** プロンプト レスポンスにオーディオ ブロックを追加します。複数回の呼び出しで複数部分のレスポンスが生成されます。
- **Result.Messages.AddEmbeddedResource(Uri, Title, Text, MimeType):** プロンプトレスポンスに埋め込みリソースブロックを追加します。複数回呼び出すと、マルチパートレスポンスが生成されます。
- **Result.Messages.Clear:** オブジェクトを再利用する前に応答をリセットします。

高度な使用例

次の例では、LLMをトリアーজワークフローに導くプロンプトを公開します。オプション引数とハンドラーでの複数ステップの指示の返し方を示しています。

```

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 サーバー | リソース

MCP では、リソースは、ファイル、データベースレコード、生成されたドキュメント、動的な API 出力など、サーバーが公開するアドレス可能なデータオブジェクトを表します。

それらは **クライアント制御** であり、クライアントがどのリソースをリクエストするか、またコンテンツをどのように解釈するかを決定できます。

主なメソッドは以下の 2 つです:

- **resources/list:** サーバーが公開するすべての利用可能なリソース定義のリストを要求します。TsgcWSAPIServer_MCP コンポーネントによって内部的に処理されます。
- **resources/read:** 特定のリソースのコンテンツおよびオプションのメタデータを取得します。サーバーが新しいリクエストを受信すると **OnMCPRequestResource** イベントが呼び出されます。リソースには通常以下が含まれます。
 - **uri:** リソース識別子。
 - **mimeType:** コンテンツタイプ (text/plain、application/json、image/png など)。
 - **data または content:** 実際のペイロード (テキストまたはバイナリ)。

また、サーバーが宣言した機能で `resources.listChanged = true` を設定している場合、リソースカタログが更新されると `notifications/resources/list_changed` のような通知を送信する場合があります。

リソースリスト

サーバーは **TsgcAI_MCP_ResourcesList** にリソースのインメモリカタログを保持しています。リソースは uri で一意であることが保証され、説明を公開し、仕様の `resources.list` メソッドへの応答で出力される JSON スキーマのような入力 URI を提供します。クライアントが `resources.list` を呼び出すと、**TsgcWSAPIServer_MCP** はリクエストを読み込み、現在のカタログをシリアル化して 200 HTTP ステータスコードで返信します。

ファイルリソースを公開するサンプルコード:

```
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
    );
}
```

リソースのリクエスト

クライアントが `resource.read` JSON-RPC リクエストを発行すると、**TsgcWSAPIServer_MCP** は強く型付けされたリクエストオブジェクト (URI リソース、名前、提供された引数を含む) をハイドレートしてから **OnMCPRequestResource** イベントを発生させます。ハンドラーはレスポンスペイロードを設定し、それが成功 HTTP ステータスコードとともにクライアントにシリアル化されて返されます。

典型的なハンドラーは次のようになります:

```
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
        );
    }
}
```

パブリックAPIサーフェス

Resources はカタログエントリ (**TsgcAI_MCP_Resource**) と **OnMCPRequestResource** 内で使用されるストリーミングヘルパーを組み合わせたものです。これらはほとんどの場合に使用するメソッドです。

TsgcAI_MCP_ResourcesList

- **AddResource(Uri, Name, Title, Description, MimeType):** 静的または動的に生成されたリソースを登録します。
- **Clear:** すべてのリソース定義を削除します。設定ファイルからリストを再構築する前に便利です。
- **Count / Items[Index]:** 列挙や診断ビューのために、カタログを公開します。

TsgcAI_MCP_Resource

- **MimeType / Title / Description:** クライアントに返されるリソースペイロードを記述する書き込み可能なプロパティ。
- **Name / Uri:** リソースエントリを再登録することなく識別子を更新するには、これらのプロパティを使用します。

TsgcAI_MCP_Response_ResourcesRead

- **Result.Contents.AddContentText(Uri, Name, Title, MimeType, Text):** テキストペイロードを呼び出し元に戻します。
- **Result.Contents.AddContentBlob(Uri, Name, Title, MimeType, Text):** テキストペイロードを呼び出し元に戻します。
- **Result.Contents.Clear:** レスポンスオブジェクトを再利用する前に発信ペイロードをリセットします。
- **Result.IsError:** リソースを生成できない場合に、応答をエラーとしてフラグ付けします。

高度な使用例

以下の例では、集約されたメトリクスを返す JSON リソースをパブリッシュします。受信 URI を検証し、リクエストに応じて異なるレスポンスを返す方法を示しています。

```
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 サーバー | ルーツ

ルートは、クライアントがサーバーに探索を許可するファイルシステムの場所をマークします。各エントリは、オプションの表示名を持つfile:// URIであり、ユーザーが共有内容を制御しながら、MCPオートメーションがプロジェクトの境界を遵守できるようにします。サーバーはroots/listを発行してカタログをクエリし、クライアントがlistChangedをアナウンスすると、notifications/roots/list_changedを受信してキャッシュビューを更新できます。ルートはクライアント所有であるため、サーバーは常にリクエスター側として動作します。ディスカバリー呼び出しを試みる前に、roots ケーパビリティがクライアントの初期化ペイロードに存在している必要があります。存在しない場合、RequestRootsList は CapabilityNotSupported を発生させます。

ルートカタログのリクエスト

TsgcWSAPIServer_MCP は roots/list をシリアルライズし、返信をコードにルーティングするヘルパーを公開しています。

- **リクエストヘルパー:** MCP セッションが roots 機能を通知したら、RequestRootsList(Session) を一度呼び出します。このメソッドは識別子を自動的に割り当て（指定しない場合）、レスポンスを相関付けられるように保留リクエストを保持します。
- **応答イベント:** OnMCPResponseRootsList(Sender, Session, Request, Response)を処理します。このイベントは、クライアントが利用可能なディレクトリで応答したときに発生します。Response.Rootsを検査するか、Response.Writeをログに記録して、JSONペイロードを調べてください。
- **変更検出:** クライアントが後で notifications/roots/list_changed を送信した場合、別の RequestRootsList を送信してサーバーサイドのキャッシュを同期させます。

サンプルコード

以下のスニペットは、セッションが確立されるとすぐにルートをリクエストし、返されたエントリをログに記録します。

```
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 サーバー | サンプリング

Samplingを使用すると、サーバーはクライアントに代わってLLM生成を実行するよう依頼できます。リクエストには、会話履歴、オプションのシステムプロンプト、モデル設定、トークン予算が記述されます。クライアントは、どのプロバイダーが使用されるかの制御を保持し、人間による確認を行い、最終的に承認された補完を返します。

クライアントは initialize 時に sampling 機能を宣言する必要があります。そうでない場合、サンプリングの試みは機能エラーで拒否されます。仕様では、ユーザーの承認を促し、会話履歴内のテキスト、画像、またはオーディオコンテンツブロックを許可します。

サンプリングリクエストの発行

TsgcWSAPIServer_MCPはヘルパーとレスポンスイベントを使用してsampling/createMessageフローを効率化します:

- **リクエストヘルパー:** **TsgcAI_MCP_Request_SamplingCreateMessage**を構築し、Params.Messages、Params.ModelPreferences、Params.SystemPrompt、およびオプションでParams.MaxTokensを設定します。RequestSamplingCreateMessage(Session, Request)に渡すと、メソッドはID（必要な場合）をスタンプして保留中の呼び出しを記録します。
- **レスポンスイベント:** OnMCPResponseSamplingCreateMessage(Sender, Session, Request, Response)を処理します。Response.Role、Response.Content、Response.Model、Response.StopReason をチェックしてワークフローを続行します。
- **エラー処理:** 拒否またはキャンセルされたリクエストは JSON-RPC エラーとして届くため、ヘルパーを try/except でラップし、ログ記録のために OnMCPException を監視してください。

サンプルコード

以下のコードスニペットはクライアントに短い要約をリクエストし、アシスタントの返信を出力します。

```
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 サーバー | エリシテーション

エリシテーションを使用すると、サーバーはワークフローの途中でクライアントにユーザーからの構造化された入力を収集するよう依頼できます。リクエストには、ユーザーフレンドリーなメッセージと、取得する必要があるフィールドを記述した制約付きJSONスキーマが含まれます。クライアントは適切なフォームをレンダリングし、ローカルで回答を検証し、ユーザーがプロンプトを承認、拒否、またはキャンセルしたかどうかを返します。クライアントは initialize 時に elicitation 機能を宣言する必要があります。スキーマは、レンダリングを簡単にし、機密データをスコープ外に保つため、プリミティブなプロパティ（文字列、数値、真偽値、または列挙型）を持つフラットなオブジェクトに意図的に限定されています。

ユーザー入力のリクエスト

TsgcWSAPIServer_MCPは、elicitation/createのヘルパーを提供し、ユーザーがインタラクションを終了したときにレスポンスイベントを発生させます:

- **リクエストヘルパー**: `TsgcAI_MCP_Request_ElicitationCreate` を作成し、`Params_Message` を設定し、`Params.RequestedSchema` を通じて期待するペイロードを記述します。`RequestElicitationCreate(Session, Request)` を呼び出してクライアントに送信します。
- **レスポンスイベント**: `OnMCPResponseElicitationCreate(Sender, Session, Request, Response)` を処理します。`Response.Action` (accept、decline、または cancel) を確認し、ユーザーが承認した場合は `Response.Content` を解析してください。
- **バリデーション**: 拒否されたリクエストは JSON-RPC エラーとして届きます。コンテンツを消費する前に必ず `Action` を確認し、ユーザーがキャンセルした場合は再度尋ねる準備をしてください。

サンプルコード

以下のコードスニペットは連絡先情報をリクエストし、返信を待機して、結果のアクションとペイロードをログに記録します。

```
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 コンポーネント。

はじめに

TsgcWSAPIClient_MCP は、sgcWebSockets ネットワークコンポーネント上に Model Context Protocol (MCP) コンシューマーを実装します。セッションネゴシエーション、JSON-RPC リクエスト/レスポンスのマーシャリング、および便利なイベントを処理するため、Delphi & Cbuilder アプリケーションが MCP 準拠のサーバーが公開するプロンプト、リソース、ツールを検出できます。

設定

- **コンポーネントのドロップ:** フォーム/データモジュールに **TsgcWSAPIClient_MCP** を配置します。
- **MCP エンドポイントを指定:** **MCPOptions.HttpOptions.URL** にプロトコルを公開するサーバーの HTTP URL を設定します (例: `https://localhost:5001/mcp`)。必要に応じて、ネストされた **TLSOptions** および **LogOptions** オブジェクトを通じて TLS とログを設定します。
- **クライアントを記述する:** 製品の *Name*、人間が読める *Title*、セマンティックな *Version* で **MCPOptions.ClientInfo** を設定します。これらの値は **initialize** ハンドシェイク中に転送されます。
- **セッションの保護:** サーバーが API キーまたはカスタム HTTP ヘッダーを期待する場合は **MCPOptions.AuthenticationOptions** を設定してください。資格情報はストリーミングアップグレードを含むすべての HTTP リクエストにアタッチされます。
- **キープアライブの調整:** **MCPOptions.HeartBeat** を使用して自動 ping サイクルを有効にし、推奨間隔 (秒単位) を定義します。コンポーネントは内部キープアライブタイマーが発火したときに内部的に **Ping** を呼び出します。
- **イベントの処理:** レスポンスのログ記録、セッションの承認/拒否、またはサーバーから返されたペイロードの後処理を行うために、後述のイベントをサブスクライブしてください。

リファレンス

- プロパティ — コンポーネントで利用可能な2つのプロパティ。
- メソッド — 公開メソッド 12 件。
- イベント — 12 個のイベントを処理できます。

MCP Client | Tools

TsgcWSAPIClient_MCP コンポーネントは、サーバーが提供するツールを列挙し、ツールを呼び出すための Model Context Protocol (MCP) で定義された JSON-RPC 呼び出しを公開します。各リクエストは Initialize が成功した後に送信される必要があります。すべてのメソッドはセッションがすでに 初期化済み状態にあることを検証するためです。

ListTools リクエスト

ListTools を呼び出して、MCPサーバーが現在公開しているツールのレジストリを取得します。コンポーネントは tools/list JSON-RPCリクエストを自動的に構築し、解析されたレスポンスとともに **OnMCPListTools** イベントを発生させます。

- **Request:** ListTools (パラメータなし)。
- **イベント:** OnMCPListTools(Sender, Request, Response)。
- **Response:** Tools キューを含む TsgcAI_MCP_Response_ToolsList。各アイテムはツールの Name、Title、Description、その InputSchema (JSONスキーマタイプと名前付きプロパティ) を公開する TsgcAI_MCP_Tool です。

キューの Count および Item[Index] ヘルパーまたは GetTool を使用してカタログを検査します。

ToolsCall リクエスト

RequestTool(Name, Arguments) を呼び出して、公開されているツールの1つを呼び出します。コンポーネントは JSON引数をリクエストペイロードにコピーし、レスポンスが解析された後に **OnMCPResponseTool** を発生させます。

- **リクエスト:** ツールの 名前とオプションの JSON 引数を含む RequestTool。
- **イベント:** OnMCPResponseTool(Sender, Request, Response)。
- **Response:** TsgcAI_MCP_Response_ToolsCall。Result.IsError、Result.StructuredContentを確認し、構造化された出力フラグメントについてResult.Contentを反復処理します。ツールレスポンスには、テキスト、画像、音声blob、埋め込みリソース、またはリソースリンクを含めることができます。

サンプルコード

次のスニペットは、ツールインベントリのリクエスト、ツールの実行、および結果の読み取り方法を示しています。

```
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 クライアント | プロンプト

MCP Client コンポーネントは、サーバーが提供するプロンプトに関する2つのJSON-RPCメソッドを提供します。利用可能なプロンプトテンプレートの一覧表示と、プロンプト実行の要求です。これらの呼び出しを行う前に、クライアントが初期化されていることを確認してください。

PromptsList リクエスト

ListPrompts は現在のプロンプトカタログをリクエストします。レスポンスを受信すると、コンポーネントは **OnMCPListPrompts** を発生させます。

- **リクエスト:** ListPrompts (パラメータなし)。
- **イベント:** OnMCPListPrompts(Sender, Request, Response)。
- **Response:** TsgcAI_MCP_Response_PromptsListはPromptsキューを公開します。各TsgcAI_MCP_Promptには、そのName、Title、Description、およびArgumentsコレクション (引数名、説明、必須フラグ) が含まれます。

PromptsGet リクエスト

RequestPrompt(Name, Arguments) は供給された JSON 引数でプロンプトテンプレートを実行します。サーバーの応答が準備できると、コンポーネントは **OnMCPResponsePrompt** を発生させます。

- **Request:** RequestPrompt (プロンプトの **Name** とオプションの JSON **Arguments**)。
- **イベント:** OnMCPResponsePrompt(Sender, Request, Response)。
- **Response:** TsgcAI_MCP_Response_PromptsGet。Result.Descriptionで概要を確認し、Result.Messagesを反復処理して各TsgcAI_MCP_Response_PromptsGet_Result_Messageを読み取ります。メッセージはポリモーフィックです。特定の派生型 (..._Text、..._Image、..._Audio、..._EmbeddedResource) を使用して、基になるコンテンツと注釈にアクセスしてください。

サンプルコード

以下の例では、プロンプトリストを読み込み、プロンプトを呼び出し、返されたメッセージを出力します。

```
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 クライアント | リソース

MCP リソース API により、クライアントはセッションが初期化された後、**TsgcWSAPIClient_MCP** コンポーネントを通じてサーバーサイドのアーティファクトを検出してその内容を取得できます。

ResourcesList リクエスト

リソースカタログを取得するには **ListResources** を使用します。コンポーネントは JSON-RPC 返信の解析後に **OnMCPListResources** を発生させます。

- リクエスト: **ListResources** (パラメーターなし)。
- イベント: **OnMCPListResources(Sender, Request, Response)**。
- レスポンス: **Resources** キューを公開する **TsgcAI_MCP_Response_ResourcesList**。すべての **TsgcAI_MCP_Resource** エントリには、リソースの **Uri**、**Name**、**Title**、**Description**、**MimeType** が含まれます。

ResourcesRead リクエスト

特定のリソースをダウンロードするには **RequestResource(Uri)** を呼び出してください。コンポーネントはデコードされたペイロードとともに **OnMCPResponseResource** を発生させます。

- リクエスト: ターゲット **Uri** を持つ **RequestResource**。
- イベント: **OnMCPResponseResource(Sender, Request, Response)**。
- **Response**: **TsgcAI_MCP_Response_ResourcesRead**。 **Result.Contents** を読み取って各 **TsgcAI_MCP_Response_Resources_ReadResult_Content** ブロックにアクセスします。各ブロックには **Uri**、**Name**、**Title**、**MimeType**、および **Text** または **Blob** データが含まれます。

サンプルコード

このスニペットはリソースを一覧表示し、単一のファイルを取得します。

```
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 クライアント | ルート

MCP 仕様は **roots** ケーパビリティを導入し、クライアントがサーバーに公開する意志があるファイルシステムの場所を説明できるようにします。各 root はオプションの表示名を伴った file:// URI です。サーバーは roots/list でカタログを照会し、サポートされている場合は選択が変更されるたびに notifications/roots/list_changed メッセージを受信します。このハンドシェイクにより、ツールはプロジェクト境界、ワークスペース、マルチリポジトリのレイアウトを尊重しながら、ユーザーがサーバーにアクセスできるものをコントロールできます。

ルートの理解

Rootsは、サーバー側の機能をスコープするガードレールとして機能します。クライアントはinitialize時にroots機能を有効にし、必要に応じてlistChangedフラグを有効にすることでサポートをアドバタイズし、動的な更新をプッシュできるようにします。roots/listから返される一般的なペイロードには、それぞれUriとNameを持つ1つ以上のエントリが含まれ、ワークスペース、リポジトリ、またはその他の整理されたフォルダーを表します。

roots/listリクエストの処理

TsgcWSAPIClient_MCP は受信 JSON-RPC メッセージを解析し、サーバーが現在のカタログをリクエストするたびに **OnMCPListRoots** を発生させます。このイベントを使用して、アプリケーションが維持するワークスペースモデルをプロトコル応答と同期します。

- **Request:** TsgcAI_MCP_Request_RootsList。メッセージにはパラメータが含まれませんが、レスポンスでミラーリングする必要があるJSON-RPC識別子を提供します。
- **イベント:** OnMCPListRoots(Sender, Request, Response)。これはコンポーネントがセッションを検証した後にトリガーされます。
- **応答:** TsgcAI_MCP_Response_RootsList。各アクセス可能な場所を説明するために Roots.AddRoot(Uri, Name) を呼び出す（または TsgcAI_MCP_Root インスタンスを手動でビルドする）。ハンドラーが返るとコンポーネントが設定されたオブジェクトをシリアル化するため、リストが現在の状態を反映していることを確認してください。診断やログには Response.Write を使用します。

アプリケーションがユーザーにワークスペースのランタイムでの追加や削除を許可する場合は、対応する notifications/roots/list_changed 通知を発生させ、サーバーがキャッシュされたカタログを更新するようにしてください。

サンプルコード

以下のスニペットは、サーバーがそれらをリクエストした際に2つのルートを開示し、送り返される JSON ペイロードをログに記録する方法を示しています。

```
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 クライアント | サンプリング

サンプリングにより MCP サーバーはプロバイダーの資格情報を共有せずに言語モデル生成をクライアントに委任できます。サーバーは会話の状態、モデルの設定、maxTokens などのオプションの制約をカプセル化した `sampling/createMessage` リクエストを発行します。クライアントは人間のレビュアーを関与させ、ガードレールを適用し、選択したモデル、生成されたコンテンツ、停止理由を含む応答を最終的に返すことができます。

サンプリングについて

initialize中にsampling機能がアドバタイズされると、サーバーはワークフローの途中で生成を要求できるようになります。リクエストには、メッセージ配列（テキスト、画像、または音声ブロック）、systemPrompt、オプションのトークン制限、およびコスト、速度、知能を数値的な優先度とヒント文字列を介してバランスさせる ModelPreferences 構造体がまとめられます。クライアントは、これらのヒントを利用可能なモデルにマッピングし、必要に応じてユーザーにプロンプトを表示し、応答する前に安全でない補完をフィルタリングする責任があります。

sampling/createMessage リクエストの処理

`TsgcWSAPIClient_MCP` は `OnMCPSamplingCreateMessage` を通じてサーバー開始の生成を表面化します。ハンドラーは提供されたコンテキストを調べ、目的のモデル（または人間のレビュアー）に呼び出しをルーティングし、レスポンスペイロードを設定する必要があります。

- **Request:** `TsgcAI_MCP_Request_SamplingCreateMessage`。Params.Messages を検査して対話を再生し、Params.ModelPreferences でヒントの優先順位付けを、Params.SystemPrompt でガードレールを確認します。
- **イベント:** `OnMCPSamplingCreateMessage(Sender, Request, Response)`。コンポーネントが MCP セッションを検証し、リクエストをデシリアライズした後に発生します。
- **レスポンス:** `TsgcAI_MCP_Response_SamplingCreateMessage`。アシスタントの Role を提供し、コンテンツオブジェクト（テキスト、画像、オーディオ、またはリソースリンク）を割り当て、オプションで具体的な Model と StopReason を記録します。Response.Write を呼び出して、ディスパッチされる前に JSON ペイロードをプレビューします。

サンプルコード

以下のスニペットは、受信メッセージ数をログに記録し、固定の応答を生成して、モデルと停止理由を注釈付けします。本番環境では、ハードコードされた内容を実際のモデル呼び出しや人間による承認ループに置き換えてください。

```
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)
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

エリシテーションにより、サーバーは MCP クライアントを通じてユーザーから構造化された入力をリクエストできます。リクエストには人間が読めるメッセージと、期待されるフィールドを定義する制約付き JSON スキーマが含まれます。クライアントはインタラクションの制御を維持し、提供されたデータを検証し、ユーザーがプロンプトを承諾、拒否、またはキャンセルしたかどうかを示すアクションを返します。

エリシテーションについて

elicitation ケーパビリティは initialize 中にネゴシエートされます。有効化されると、サーバーはプリミティブ型（文字列、数値/整数、ブール値、およびオプションの表示名を持つ列挙）に限定されたフラットオブジェクトスキーマを埋め込んだ elicitation/create 呼び出しを送信できます。各プロパティには表示メタデータ、長さや範囲の制約、サポートされるフォーマット（メール、URI、日付、日時）を指定できます。ユーザーには常に誰が情報を要求しているかを知らせ、ペイロードを編集する機会を与え、リクエストを拒否またはキャンセルできるようにする必要があります。

elicitation/create リクエストの処理

TsgcWSAPIClient_MCP はサーバーが追加の入力を要求するたびに **OnMCPElicitationCreate** を発生させます。このイベントにより、プロンプトメッセージ、要求されたスキーマ、およびユーザーの決定に基づいて設定できる空のレスポンスオブジェクトにアクセスできます。

- **リクエスト**： **TsgcAI_MCP_Request_ElicitationCreate**。 **Params.Message** を検査してプロンプトを表示し、 **Params.RequestedSchema.Properties** で UI コントロールまたはバリデーションを構築します。
- **イベント**： **OnMCPElicitationCreate(Sender, Request, Response)**。 クライアントが JSON-RPC ペイロードをデシリアライズした後にトリガーされます。
- **Response**: **TsgcAI_MCP_Response_ElicitationCreate**。 **Action** を **accept**、**decline**、または **cancel** に設定します。受け入れる場合は、収集された値を **Content (TsgcJSON ヘルパー)** に書き込んでリクエストされたスキーマと一致させます。 **Response.Write** を使用して最終 JSON ドキュメントを確認してください。

サンプルコード

以下の例は、GitHub ハンドルをユーザーに求めます。オペレーターが値を提供した場合、レスポンスが受け入れられ、フィールドが返されます。そうでない場合、リクエストは拒否されます。

```
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

OpenAIは、人類全体に利益をもたらす方法で人工知能（AI）を開発し、方向づけることを目指す民間研究所です。OpenAIは次のプロジェクトを開発してきました:

- **GPT-3:** この強力な言語モデルは他の OpenAI 製品の基盤として機能します。人間が生成したテキストを分析して、同様のテキストを自ら生成することを学習します。
- **DALL-E および DALL-E 2:** これらの生成 AI プラットフォームは、ユーザーが生成したい画像のテキストベースの説明を分析し、説明通りに画像を生成できます。
- **CLIP:** CLIPは、視覚情報とそれに関連するテキストを合成して、それらの視覚情報を最も正確に記述する可能な限り最良のキャプションを予測するニューラルネットワークです。複数のタイプのデータ(画像とテキストの両方)から学習する能力があるため、マルチモーダルAIとして分類できます。
- **ChatGPT:** ChatGPT は現在、人間らしいテキストを生成しユーザーの質問に答えるために設計された最も高度な AI チャットボットです。大規模なデータセットでトレーニングされており、人間のように答えやレスポンスを生成できます。
- **Codex:** Codexは、ソフトウェア開発者がコーディングプロセスを簡素化できるよう支援するために、さまざまなプログラミング言語の数十億行のコードで学習されました。GPT-3技術を基盤としていますが、テキストを生成する代わりにコードを生成します。
- **Whisper:** Whisper は自動音声認識（ASR）ツールです。技術的な言語や様々なアクセントを含む約100の異なる言語で音声を認識、文字起こし、翻訳するために多量の音声データで訓練されています。

OpenAI API

OpenAI API は、自然言語、コード、または画像の理解や生成に関わる事実上あらゆるタスクに適用できます。OpenAI はさまざまなタスクに適した異なるレベルの性能を持つモデルのスペクトルを提供し、独自のカスタムモデルをファインチューニングする機能も提供しています。これらのモデルはコンテンツ生成からセマンティック検索、分類まであらゆることに使用できます。

最も一般的な使用方法

- 補完
 - [OpenAI 補完の使用例](#)
- チャット
 - [OpenAI チャットの使用例](#)
- 編集
 - [OpenAI 編集の使用例](#)
- オーディオ
 - [OpenAI 文字起こし & 翻訳の例](#)
- モデレーション
 - [OpenAI モデレーションの例](#)
- RealTime
 - [OpenAI RealTime の例](#)
- レスポンス
 - [OpenAI レスポンスの使用例](#)
- スピーチ
 - [OpenAI スピーチの使用例](#)
- ファインチューニング
 - [OpenAI ファインチューニング 使用例](#)
- Batch
 - [OpenAI Batch Examples](#)
- アップロード

- [OpenAI アップロード 使用例](#)

設定

OpenAI

OpenAI API は認証に API キーを使用します。リクエストで使用する API キーを取得するには、[API キー](#)ページをご覧ください。

API キーは秘密です！他の人と共有したり、クライアント側のコード（ブラウザー、アプリ）に公開したりしないでください。本番環境のリクエストは、環境変数またはキー管理サービスから API キーを安全に読み込める独自のバックエンドサーバー経由でルーティングする必要があります。

この **API Key** はコンポーネントの **OpenAIOptions.ApiKey** プロパティに設定する必要があります。オプションとして、複数の組織に属するユーザーは、アカウントが組織に属している場合は **OpenAIOptions.Organization** プロパティに組織を設定できます。

API キーが設定されたら、以下に OpenAI API と連携するために利用可能な関数の一覧を示します。

Azure

このクライアントは Microsoft Azure OpenAI サービスをサポートしているため、Azure アカウントを使用して Azure OpenAI API とやり取りすることもできます。クライアントを Azure で動作するように設定するには、以下の手順に従ってください：

1. プロパティ **OpenAIOptions.Provider** = `oapvAzure` を設定してください。
2. **ResourceName** と **DeploymentId** の値を設定します（これらの値は Azure アカウントで確認できます）
 1. **OpenAIOptions.AzureOptions.ResourceName** = <リソース名>。
 2. **OpenAIOptions.AzureOptions.DeploymentId** = <デプロイメント ID>。
3. Azure アカウントの API キーを設定します
 1. **OpenAIOptions.ApiKey** = <azure api key>。

すべての OpenAI メソッドが Azure でサポートされているわけではありません。現在サポートされているメソッドは以下のみです：

1. 補完
2. チャット補完

プロパティ

OpenAIOptions

- **ApiKey**: OpenAI API で認証するためのAPIキー。
- **Organization**: 複数の組織に属するユーザー向けのオプションの組織ID。
- **Provider**: プロバイダーを選択します: `oapvOpenAI`（デフォルト）、またはMicrosoft Azure OpenAI Services用の`oapvAzure`。
- **AzureOptions**: Azure OpenAI Services の設定。
 - **ResourceName**: Azure リソース名。
 - **DeploymentId**: Azure デプロイメント ID。
 - **APIVersion**: Azure API バージョン。
- **HttpOptions**: HTTP 接続設定。
 - **ReadTimeout**: HTTP レスポンスを読み取るためのタイムアウト（ミリ秒）。デフォルトは 0（タイムアウトなし）。
- **LogOptions**: ログ設定。
 - **Enabled**: True の場合、HTTP リクエストとレスポンスがファイルに記録されます。
 - **FileName**: ログ出力が書き込まれるファイルパス。
- **RetryOptions**: 失敗したAPIリクエストの自動リトライ設定。
 - **Enabled**: True の場合、失敗したリクエストは自動的にリトライされます。
 - **Retries**: リトライの最大試行回数。
 - **Wait**: リトライ試行間の待機時間（ミリ秒単位）。

モデル

API で利用可能なさまざまなモデルを一覧表示して説明します。

- **GetModels:** 現在利用可能なモデルの一覧と、オーナーや可用性などの各モデルに関する基本情報を返します。
- **GetModel:** 所有者や権限など、モデルの基本情報を提供するモデルインスタンスを取得します。
 - **Model:** このリクエストに使用するモデルの ID

補完

プロンプトが与えられると、モデルは1つ以上の予測された補完を返し、各位置での代替トークンの確率も返すことができます。

- **CreateCompletion:** 提供されたプロンプトとパラメータの補完を作成します
 - **Model:** 使用するモデルの ID。List models API を使用してすべての利用可能なモデルを確認するか、モデルの概要で説明を参照してください。
 - **Prompt:** 補完を生成するためのプロンプト。

チャット

チャット会話が与えられると、モデルはチャット補完レスポンスを返します。

- **Model:** 使用するモデルの ID。Chat API がサポートするすべてのモデルのリストを取得するには GetModels を呼び出します。
- **Message:** チャット補完を生成するためのメッセージ。
- **Role:** デフォルトでは user。他のオプション: system、assistant。

編集

プロンプトと指示が与えられると、モデルはプロンプトの編集済みバージョンを返します。

- **CreateEdit:** 指定された入力、指示、パラメータに基づいて新しい編集を作成します。
 - **Model:** 使用するモデルの ID。このエンドポイントでは text-davinci-edit-001 または code-davinci-edit-001 モデルを使用できます。
 - **Instruction:** モデルにプロンプトの編集方法を指示する説明。
 - **Input:** (オプション) 編集の開始点として使用する入力テキスト。

画像

プロンプトおよび/または入力画像が与えられると、モデルは新しい画像を生成します。

- **CreateImage:** プロンプトに基づいて画像を作成します。
 - **Prompt:** 目的の画像のテキスト説明。最大長は1000文字です。
- **CreateImageEdit:** 元の画像とプロンプトに基づいて編集または拡張された画像を作成します。
 - **Image:** 編集する画像。有効な PNG ファイルで 4MB 未満の正方形である必要があります。マスクが提供されない場合、画像にはマスクとして使用される透明度が必要です。
 - **Prompt:** 目的の画像のテキスト説明。最大長は 1000 文字です。
- **CreateImageVariations:** 指定した画像のバリエーションを作成します。
 - **Image:** バリエーションのベースとして使用する画像です。有効なPNGファイルで、4MB未満かつ正方形である必要があります。

エンベディング

機械学習モデルやアルゴリズムで簡単に利用できる、指定された入力のベクトル表現を取得します。

- **CreateEmbeddings:** 入力テキストを表すembeddingベクトルを作成します。
 - **Model:** 使用するモデルのID。
 - **Input:** embeddingsを取得する入力テキスト。

音声

音声をテキストに変換します。

- **CreateTranscriptionFromFile:** ファイル名から入力言語の音声をテキストに転写します
 - **Model:** 使用するモデルの ID。現在は whisper-1 のみが利用可能です。
 - **Filename:** 転写する音声ファイル（対応形式: mp3、mp4、mpeg、mpga、m4a、wav、または webm）。
- **CreateTranscription:** X 秒間音声を録音して文字起こしします。
 - **Model:** 使用するモデルの ID。現在は whisper-1 のみ利用可能です。
 - **Time:** ミリ秒単位の時間。デフォルトは 10 秒です。
- **CreateTranslationFromFile:** 音声を英語に翻訳します。
 - **Model:** 使用するモデルのID。現在whisper-1のみが利用可能です。
 - **Filename:** 翻訳する音声ファイル。次の形式のいずれか: mp3、mp4、mpeg、mpga、m4a、wav、またはwebm。
- **CreateTranslation:** X 秒間オーディオを録音して翻訳します。
 - **Model:** 使用するモデルの ID。現在は whisper-1 のみ利用可能です。
 - **Time:** ミリ秒単位の時間。デフォルトは 10 秒です。

ファイル

ファイルは、ファインチューニングなどの機能で使用するドキュメントのアップロードに使用されます。

- **ListFiles:** ユーザーの組織に属するファイルのリストを返します。
- **UploadFile:** さまざまなエンドポイント/機能で使用するドキュメントを含むファイルをアップロードします。現在、1つの組織がアップロードするすべてのファイルの合計サイズは最大 1 GB です。
 - **Filename:** アップロードする JSON Lines ファイルの名前。目的が "fine-tune" に設定されている場合、各行はトレーニング例を表す "prompt" と "completion" フィールドを持つ JSON レコードです。
 - **Purpose:** アップロードされたドキュメントの目的。Fine-tuning には "fine-tune" を使用します。
- **DeleteFile:** ファイルを削除します。
 - **FileId:** このリクエストに使用するファイルの ID
- **RetrieveFile:** 指定したファイルに関する情報を返します。
 - **FileId:** この要求に使用するファイルのIDです
- **RetrieveFileContent:** 指定されたファイルの内容を返します
 - **FileId:** このリクエストで使用するファイルの ID。

ファインチューニング

特定のトレーニングデータにモデルをカスタマイズするためのファインチューニングジョブを管理します。

- **CreateFineTune:** 指定されたデータセットから指定されたモデルをファインチューニングするジョブを作成します。レスポンスには、ジョブステータスや完了後のファインチューニングされたモデルの名前を含む、エンキューされたジョブの詳細が含まれます。
 - **TrainingFile:** トレーニングデータを含むアップロード済みファイルの ID。
- **ListFineTunes:** 組織のファインチューニングジョブを一覧表示します。
- **RetrieveFineTune:** ファインチューニングジョブに関する情報を取得します。
 - **FineTuneId:** ファインチューニングジョブのID
- **CancelFineTune:** ファインチューニングジョブを即座にキャンセルします。
 - **FineTuneId:** ファインチューニングジョブの ID
- **ListFineTuneEvents:** ファインチューニングジョブの詳細なステータス更新を取得します。
 - **FineTuneId:** ファインチューニングジョブの ID。
- **DeleteFineTuneModel:** ファインチューニングされたモデルを削除します。組織でOwnerロールを持っている必要があります。
 - **Model:** 削除するモデル。

モデレーション

入力テキストに対して、モデルが OpenAI のコンテンツポリシーに違反するものとして分類するかどうかを出力します。

- **CreateModeration:** テキストが OpenAI のコンテンツポリシーに違反するかどうかを分類します。
 - **Input:** 分類する入力テキスト

RealTime

OpenAI Realtime API は、音声間の会話体験やリアルタイムの文字起こしを含む、低遅延のマルチモーダルインタラクションを実現します。

- **文字起こし:** Realtime API をマイク入力またはファイルからの入力による文字起こし専用のユースケースに使用できます。たとえば、リアルタイムで字幕やトランスクリプトを生成するために使用できます。文字起こし専用モードでは、モデルはレスポンスを生成しません。

アシスタント

モデルを呼び出してタスクを実行するツールを使用できる AI アシスタントを構築します。

- **CreateAssistant:** モデルと指示を使用してアシスタントを作成します。
- **ListAssistants:** アシスタントのリストを返します。
- **RetrieveAssistant:** IDによってアシスタントを取得します。
- **ModifyAssistant:** 既存のアシスタントを変更します。
- **DeleteAssistant:** アシスタントを削除します。

Threads

スレッドは会話セッションを表します。メッセージはスレッドに追加され、runによって処理されます。

- **CreateThread:** 新しいスレッドを作成します。
- **RetrieveThread:** ID でスレッドを取得します。
- **ModifyThread:** スレッドを変更します。
- **DeleteThread:** スレッドを削除します。

スレッドメッセージ

メッセージはスレッドに追加され、会話の内容を含みます。

- **CreateMessage:** スレッド内にメッセージを作成します。
 - **ThreadId:** スレッドのID。
- **ListMessages:** 指定されたthreadのメッセージのリストを返します。
- **RetrieveMessage:** スレッドとメッセージ ID でメッセージを取得します。
- **ModifyMessage:** メッセージを変更します。
- **DeleteMessage:** スレッドからメッセージを削除します。

Runs

実行は Assistant を使用した Thread 上の実行を表します。Assistant はその設定と Thread のメッセージを使用して、モデルとツールを呼び出してタスクを実行します。

- **CreateRun:** 指定したアシスタントでスレッドのランを作成します。
- **ListRuns:** スレッドに属するランの一覧を返します。
- **RetrieveRun:** スレッドと実行 ID でランを取得します。
- **ModifyRun:** ランを変更します。
- **SubmitToolOutputsToRun:** アクションが必要な Run のツール出力を送信します。
- **CancelRun:** 進行中の Run をキャンセルします。

Run Steps

実行ステップは、実行中に実行される個々のステップを表します。

- **ListRunSteps:** ランに属するランステップのリストを返します。
- **RetrieveRunSteps:** スレッド、実行、ステップ ID によって実行ステップを取得します。

Vector Stores

ベクトルストアは、検索拡張生成（RAG）のために埋め込みを使用してファイルを保存・検索するために使用されます。

- **CreateVectorStore:** ベクトルストアを作成します。
- **ListVectorStores:** ベクトルストアのリストを返します。
- **RetrieveVectorStore:** IDによってベクトルストアを取得します。
- **ModifyVectorStore:** ベクトルストアを変更します。
- **DeleteVectorStore:** ベクトルストアを削除します。

ベクトルストアファイル

ベクトルストア内のファイルを管理します。

- **CreateVectorStoreFile:** ファイルをベクトルストアにアタッチします。
- **ListVectorStoreFiles:** ベクトルストア内のファイルのリストを返します。
- **RetrieveVectorStoreFile:** ベクトルストアファイルを取得します。
- **DeleteVectorStoreFile:** ベクトルストアからファイルを削除します。

Vector Store File Batches

ベクトルストアへのファイル追加のバッチ操作。

- **CreateVectorStoreFileBatch:** ベクトルストアに複数のファイルを追加するバッチを作成します。
- **RetrieveVectorStoreFileBatch:** ID によってファイルバッチを取得します。
- **CancelVectorStoreFileBatch:** 進行中のファイルバッチをキャンセルします。
- **ListVectorStoreFilesBatch:** バッチ内のファイルを一覧表示します。

音声

テキスト読み上げモデルを使用してテキストから音声を生成します。

- **CreateSpeech:** 入力テキストから音声を生成します。音声データを文字列として返すか、レスポンスストリームに書き込みます。
 - **Model:** 使用するTTSモデル（例：tts-1、tts-1-hd）。
 - **Input:** 音声を生成するためのテキスト。
 - **Voice:** 使用する音声（alloy、echo、fable、onyx、nova、shimmer）。

ファインチューニングジョブ

カスタマイズされたモデルを作成するためのファインチューニングジョブを管理します。これはレガシーの Fine-Tunes エンドポイントを置き換える新しいファインチューニング API です。

- **CreateFineTuningJob:** トレーニングファイルからファインチューニングジョブを作成します。
- **ListFineTuningJobs:** 組織のファインチューニングジョブを一覧表示します。
- **RetrieveFineTuningJob:** ファインチューニングジョブに関する情報を取得します。
- **CancelFineTuningJob:** ファインチューニングジョブをキャンセルします。

- **ListFineTuningJobEvents:** ファインチューニングジョブのステータス更新をリストします。
- **ListFineTuningJobCheckpoints:** ファインチューニングジョブのチェックポイントを一覧表示します。

レスポンス

モデルレスポンスを作成および管理します。Responses API はマルチターン会話、ツール使用、および構造化出力をサポートします。

- **CreateResponse:** モデルレスポンスを作成します。
- **RetrieveResponse:** IDでレスポンスを取得します。
- **DeleteResponse:** レスポンスを削除します。
- **CancelResponse:** 進行中のレスポンスをキャンセルします。
- **ListInputItems:** レスポンスの入力アイテムを一覧表示します。

バッチ

低コストで非同期処理するためのバッチ API リクエストを作成および管理します。

- **CreateBatch:** バッチ ジョブを作成します。
- **RetrieveBatch:** IDでバッチを取得します。
- **ListBatches:** すべてのバッチを一覧表示します。
- **CancelBatch:** 進行中のバッチをキャンセルします。

アップロード

大きなファイルをパートに分けてアップロードします。標準のアップロードサイズ制限を超えるファイルに便利です。

- **CreateUpload:** アップロードセッションを作成します。
- **AddUploadPart:** アップロードにパートを追加します。
- **CompleteUpload:** マルチパートアップロードを完了し、ファイルを作成します。
- **CancelUpload:** 進行中のアップロードをキャンセルします。

TsgcHTTP_API_OpenAI

チャット補完、画像、音声 (TTS/STT)、エンベディング、ファイル、ファインチューニング、モデレーション、アシスタント、スレッド、ラン、メッセージ、ベクターストアをカバーする OpenAI REST API クライアント。

はじめに

チャット補完、画像、音声 (TTS/STT)、エンベディング、ファイル、ファインチューニング、モデレーション、アシスタント、スレッド、ラン、メッセージ、ベクターストアをカバーする OpenAI REST API クライアント。

はじめに

チャット補完、画像、音声 (TTS/STT)、エンベディング、ファイル、ファインチューニング、モデレーション、アシスタント、スレッド、ラン、メッセージ、ベクターストアをカバーする OpenAI REST API クライアント。

OpenAIOptions プロパティを API キーおよびオプションの HTTP/リトライ/ログ設定で構成し、`CreateMessage/``CreateChatCompletion`などの型付きメソッドをリクエストオブジェクトとともに呼び出すと、型付きレスポンスクラスを受け取ります。

プロバイダーの背景については、[OpenAI](#) ガイドを参照してください。

リファレンス

- プロパティ — コンポーネントで利用可能な5個のプロパティ。
- メソッド — 78個のパブリックメソッド。
- イベント — 処理できる2つのイベント。

OpenAI | 補完

プロンプトが与えられると、モデルは1つ以上の予測される補完を返し、各位置で代替トークンの確率を返すこともできます。

簡単な例

text-davinci-003 モデルを使用して予測補完を取得します。

```
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"));
```

高度な例

text-davinci-003 モデルを使用して、よりランダムな出力で予測補完を取得し、各プロンプトに対して2つの補完を生成します。

```
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

チャット会話が与えられると、モデルはチャット補完レスポンスを返します。

簡単な例

Hello メッセージを送信して ChatGPT と対話します。

```
TsgcHTTP_API_OpenAI *OpenAI = new TsgcHTTP_API_OpenAI(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";
ShowMessage(OpenAI->_CreateChatCompletion("gpt-3.5-turbo", "Hello!"));
```

高度な例

gpt-3-5 モデルを使用して、よりランダムな出力でチャットし、各プロンプトに対して 2 つの補完を生成します。

```
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 | 編集

プロンプトと指示が与えられると、モデルはプロンプトの編集版を返します。

簡単な例

OpenAI にプロンプトのスペルミスを修正させます。

```
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?"));
```

高度な例

OpenAI に、よりランダムな出力でプロンプトのスペルミスを修正し、各プロンプトに対して 2 つの補完を生成するよう指示します。

```
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 | 音声

Create Transcription

音声を入力言語に文字起こしします。

```
TsgcHTTP_API_OpenAI *OpenAI = new TsgcHTTP_API_OpenAI(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";

ShowMessage(OpenAI->_CreateTranscriptionFromFile("whisper-1", "c:\\media\\audio.mp3"));
```

翻訳の作成

音声を英語に翻訳します。

```
TsgcHTTP_API_OpenAI *OpenAI = new TsgcHTTP_API_OpenAI(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";

ShowMessage(OpenAI->_CreateTranslationFromFile("whisper-1", "c:\\media\\audio.mp3"));
```


OpenAI | モデレーション

入力テキストが与えられると、モデルが OpenAI のコンテンツポリシーに違反していると分類するかどうかを出力します。

簡単な例

以下のテキストをモデレートしてください

```
TsgcHTTP_API_OpenAI *OpenAI = new TsgcHTTP_API_OpenAI(NULL);
OpenAI->OpenAIOptions->ApiKey = "API_KEY";

ShowMessage(OpenAI->_CreateModeration("I want to kill them."));
```

高度な例

モデルを選択して以下のテキストをモデレートします。

```
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

マイクまたはファイルからの入力を使用して、文字起こしのみのユースケースに Realtime API を使用できます。例えば、リアルタイムで字幕やトランスクリプトを生成するために使用できます。文字起こしのみモードでは、モデルはレスポンスを生成しません。

トランスクリプション用のリアルタイム API を使用するには、WebSocket で接続してトランスクリプションセッションを作成する必要があります。コンポーネント [TsgcWSAPI_OpenAI](#) と [TsgcWebSocketClient](#) を使用して新しいトランスクリプションセッションを開始します。

以下に OpenAI API を使用したリアルタイム文字起こしの使用例を示します。

```
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);
}
```

オーディオを手動で送信

コンポーネントはオーディオファイルを手動で送信できます。メソッド `AppendInputAudioBuffer` を使用して `TStream` としてオーディオを渡すことができます。オーディオフォーマットは 24 kHz モノ PCM (24000 のレートのみサポート) である必要があります。

OpenAI | Responses

Responses API は OpenAI モデルと対話するためのコア API です。テキストと画像の入力、ツール使用、ストリーミング、構造化出力をサポートします。

簡単な例

Responses APIを使用してレスポンスを作成します。

```
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?"));
```

メソッド

- **CreateResponse:** モデルと入力からモデルレスポンスを作成します。
 - **Model:** 使用するモデルのID (例: gpt-4o、gpt-4o-mini)。
 - **Input:** モデルへのテキストまたは構造化入力。
- **RetrieveResponse:** 以前に作成されたレスポンスを ID で取得します。
 - **ResponseId:** 取得するレスポンスの ID。
- **DeleteResponse:** ID でレスポンスを削除します。
 - **ResponseId:** 削除するレスポンスの ID。
- **CancelResponse:** 進行中のレスポンスをキャンセルします。
 - **ResponseId:** キャンセルするレスポンスの ID。
- **ListInputItems:** 指定されたレスポンスの入力アイテムの一覧を返します。
 - **ResponseId:** レスポンスの ID。

OpenAI | 音声

Text-to-Speech (TTS) API を使用してテキストから音声を生成します。複数の音声と出力形式をサポートしています。

簡単な例

テキストから音声を生成してファイルストリームに保存します。

```
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;
}
```

メソッド

- **CreateSpeech**：入力テキストからオーディオを生成します。
 - **Model**：使用するモデルの ID (tts-1 または tts-1-hd)。
 - **Input**：オーディオを生成するテキスト。最大 4096 文字。
 - **Voice**：使用する音声 (alloy、echo、fable、onyx、nova、shimmer)。
 - **ResponseStream**：オーディオデータが書き込まれるストリーム。
 - **ResponseFormat**：（オプション）オーディオ形式：mp3（デフォルト）、opus、aac、または flac。
 - **Speed**：（オプション）生成されるオーディオの速度（0.25 ～ 4.0、デフォルト 1.0）。

OpenAI | ファインチューニング

ファインチューニングジョブを管理して、特定のトレーニングデータに合わせてモデルを調整します。ファインチューニングにより、より高品質な結果、より多くのサンプルでのトレーニング能力、コスト削減など、モデルからより多くの成果を得ることができます。

簡単な例

ファインチューニングジョブを作成し、既存のジョブを一覧表示します。

```
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());
```

メソッド

- **CreateFineTuningJob:** 指定されたデータセットから新しいモデルを作成するプロセスを開始するファインチューニングジョブを作成します。
 - **Model:** ファインチューニングするモデルの名前。
 - **TrainingFile:** トレーニングデータを含むアップロードされたファイルの ID。
- **ListFineTuningJobs:** 組織のファインチューニングジョブを一覧表示します。
- **RetrieveFineTuningJob:** ファインチューニングジョブの情報を取得します。
 - **JobId:** ファインチューニングジョブの ID。
- **CancelFineTuningJob:** ファインチューニングジョブを即座にキャンセルします。
 - **JobId:** キャンセルするファインチューニングジョブの ID。
- **ListFineTuningJobEvents:** ファインチューニングジョブのステータス更新を取得します。
 - **JobId:** ファインチューニングジョブの ID。
- **ListFineTuningJobCheckpoints:** ファインチューニングジョブのチェックポイントを一覧表示します。
 - **JobId:** ファインチューニングジョブの ID。

OpenAI | バッチ

API リクエストの大きなバッチを非同期で実行するために作成します。Batch API は 50% の割引で 24 時間以内に補完を返します。

簡単な例

バッチジョブを作成し、そのステータスを取得します。

```
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());
```

メソッド

- **CreateBatch:** アップロードされたリクエストファイルからバッチを作成して実行します。
 - **InputFileId:** バッチのリクエストを含むアップロードされたファイルのID。
 - **Endpoint:** バッチ内のすべてのリクエストに使用するエンドポイント（例: /v1/chat/completions）。
 - **CompletionWindow:** （オプション）バッチを処理する時間枠。デフォルトは24時間。
- **RetrieveBatch:** IDでバッチを取得します。
 - **BatchId:** 取得するバッチのID。
- **ListBatches:** 組織のバッチを一覧表示します。
- **CancelBatch:** 進行中のバッチをキャンセルします。
 - **BatchId:** キャンセルするバッチのID。

OpenAI | アップロード

ファイルをパーツに分けてアップロードします。Uploads APIを使用すると、512 MBを超えるファイルをパーツに分割して各パーツを個別にアップロードできます。

簡単な例

アップロードを作成し、パートを追加して完了します。

```
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\"]"));
```

メソッド

- **CreateUpload:** マルチパートアップロードを開始するための中間 Upload オブジェクトを作成します。
 - **Filename:** アップロードするファイルの名前。
 - **Purpose:** アップロードされたファイルの使用目的 (例: fine-tune)。
 - **Bytes:** ファイルのバイト数。
 - **MimeType:** ファイルの MIME タイプ (例: application/jsonl)。
- **AddUploadPart:** Upload オブジェクトにパートを追加します。
 - **UploadId:** アップロードの ID。
 - **Filename:** このパートのバイトのチャンク。
- **CompleteUpload:** アップロードを完了します。アップロードされたファイルは、完了後にのみ使用可能になります。
 - **UploadId:** アップロードのIDです。
 - **PartIds:** JSON配列文字列としての、順序付けられたパートIDのリストです。
- **CancelUpload:** アップロードをキャンセルします。キャンセル後はパーツを追加できません。
 - **UploadId:** キャンセルするアップロードの ID。

OpenAI アプリケーション

概要

OpenAI APIを使用すると、さまざまなアプリケーションを構築できます。以下にいくつかの例を示します:

- **チャットボットおよびバーチャルアシスタント:** 人間と自然で人間らしい方法で会話できるアプリケーション。カスタマーサポート、問い合わせへの対応、ウェブサイトやモバイルアプリでの情報提供などに使用できます。
- **コンテンツ生成:** 記事、ブログ投稿、レポートなど人間のようなテキストを生成できるアプリケーション。例えば、GPT-3 はソーシャルメディアのコンテンツ作成の自動化、コードの生成、または SEO に適したコンテンツの作成に使用できます。
- **翻訳サービス:** テキストをある言語から別の言語に翻訳できるアプリケーション。
- **チュータリングと教育:** AI はパーソナライズされた学習体験の作成、宿題の支援、または複雑な概念をシンプルな言葉で説明するために使用できます。
- **Games:** OpenAIは、特に会話型キャラクターや複雑な物語を含む、没入型でインタラクティブなゲームの作成に使用できます。
- **感情分析:** テキストデータに表現された感情を分析および分類することは、市場調査、ブランドモニタリング、顧客フィードバックの理解に役立ちます。
- **パーソナライズされた推薦:** ユーザーの過去の行動と好みに基づいて、AI は製品、サービス、またはコンテンツのパーソナライズされた推薦を生成できます。
- **テキスト補完:** メールクライアントやワープロソフトウェアでのユーザーの文章補完や文章作成支援。
- **音声認識:** 話し言葉を文字に変換します。文字起こしサービス、音声アシスタントなどに役立ちます。
- **医療・法律アドバイス:** 専門的なアドバイスの代替にはなりませんが、AIモデルは与えられた入力に基づいて予備的なガイダンスや提案を提供できます。

コンポーネント

以下に、利用可能なコンポーネントのリストとそれらの簡単な説明を示します。

- **TsgcAIOpenAIAssistant:** アシスタントは指示を持ち、モデル、ツール、ファイルを活用してユーザーのクエリに応答できます。
- **TsgcAIOpenAIChatBot:** 音声を聞いてOpenAI Whisper APIを使用してテキストに変換するChatBotです。このテキストはChatCompletion APIに送信され、OpenAIからのレスポンスが提供され、このレスポンスがテキストから音声に変換されます。
- **TsgcAIOpenAITranslator:** 任意の言語の音声を英語に翻訳し、利用可能な TextToSpeech コンポーネントを使用して翻訳を聴くことができる翻訳アプリケーション。
- **TsgcAIOpenAIEmbeddings:** このコンポーネントは、カスタマイズされたデータでAIアプリケーションを構築するために使用されます。例: 製品データを持つチャットボット。次のデータベースがサポートされています。
 - **TsgcAIDatabaseVectorPinecone:** Pinecone Database Vectorをサポートします。

以下のコンポーネントは、マイクからの音声キャプチャ、音声ファイルの再生、テキストから音声への変換に使用されます。

- **TsgcAudioRecorderMCI:** (Windowsのみ)このコンポーネントを使用すると、マイクにアクセスして音声を wave ファイルに変換できます。
- **TsgcAudioPlayerMCI:** (Windows のみ) mp3 ファイルを再生できるコンポーネント。
- **TsgcTextToSpeechSystem:** (Windowsのみ) 外部mp3ファイルを使用せずにテキストを音声に変換します。
- **TsgcTextToSpeechGoogle:** 利用可能なGoogle Cloud音声を使用してテキストを音声に変換します。
- **TsgcTextToSpeechAmazon:** Amazon AWS で利用可能な音声を使用してテキストを音声に変換します。

TsgcAIOpenAIAssistant

Assistants APIを使用すると、独自のアプリケーション内にAIアシスタントを構築できます。

はじめに

Assistants API を使用すると、独自のアプリケーション内に AI アシスタントを構築できます。アシスタントには指示があり、モデル、ツール、ファイルを活用してユーザーのクエリに応答できます。Assistants API は現在、コードインタープリター、ファイル検索、関数呼び出しの 3 種類のツールをサポートしています。

概要

Assistants API の典型的な統合フローは以下のとおりです：

- ・ カスタム指示を定義してモデルを選択することでアシスタントを作成します。役立つ場合は、ファイルを追加してコードインタープリタ、ファイル検索、関数呼び出しなどのツールを有効にします。
- ・ ユーザーが会話を開始したときにスレッドを作成します。
- ・ ユーザーが質問するたびに、Threadへメッセージを追加します。
- ・ モデルとツールを呼び出して応答を生成するために、Thread上でAssistantを実行します。

ステップ 1: アシスタントを作成する

アシスタントは、モデル、指示、ツールなどのいくつかのパラメータを使用してユーザーのメッセージに返答するよう設定できるエンティティを表します。

```
// 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();
```

ステップ2: スレッドの作成

Thread はユーザーと 1 つまたは複数の Assistant 間の会話を表します。ユーザー (または AI アプリケーション) が Assistant との会話を開始するときに Thread を作成できます。

```
oThread = oAssistant->CreateThread();
```

ステップ3: スレッドにメッセージを追加して実行する

ユーザーまたはアプリケーションが作成したメッセージの内容は、スレッドにメッセージオブジェクトとして追加されます。メッセージにはテキストとファイルの両方を含めることができます。スレッドに追加できるメッセージ数に制限はありません。モデルのコンテキストウィンドウに収まらないコンテキストはスマートに切り捨てられます。

すべてのユーザーメッセージがスレッドに追加されたら、任意のアシスタントでスレッドを実行できます。実行を作成すると、アシスタントに関連付けられたモデルとツールを使用してレスポンスが生成されます。これらのレスポンスはアシスタントメッセージとしてスレッドに追加されます。

```
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"));
                }
            }
        }
    }
}
```

リファレンス

- プロパティ — コンポーネントで利用可能なプロパティが5つあります。
- メソッド — パブリックメソッドが9個あります。
- イベント — 処理できる8つのイベント。

TsgcAIOpenAIAssistant | ファイル検索

ファイル検索は、独自製品情報やユーザーが提供するドキュメントなどの外部ソースからの知識を統合することで、アシスタントの機能を強化します。仕組みは次のとおりです：

- **ドキュメント処理:** OpenAI はドキュメントを自動的に処理し、管理しやすい情報の断片 (「チャンク」) に分割します。
- **埋め込みの作成:** これらのチャンクはコンテンツの意味とコンテキストを捉える数学的表現である埋め込みに変換されます。
- **ストレージと検索:** システムはこれらの埋め込みを保存し、ベクトル検索 (意味的に類似したコンテンツを見つける) とキーワード検索 (完全一致を見つける) の両方を可能にします。

この組み合わせにより、アシスタントは関連するコンテンツを取得・活用して、ユーザーの質問に対して正確で詳しい回答を提供できます。

ステップ1: File Searchを有効にして新しいアシスタントを作成する

アシスタントの tools パラメータで file_search を有効にして新しいアシスタントを作成します。file_search ツールが有効になると、モデルはユーザーメッセージに基づいてコンテンツを取得するタイミングを決定します。

```
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;
```

ステップ 2: ファイルをアップロードして Vector Store に追加する

ファイルにアクセスするには、file_search ツールが Vector Store オブジェクトを使用します。ファイルをアップロードし、それらを格納するための Vector Store を作成してください。

```
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
    }
}

```

ステップ 3: 実行を作成して出力を確認する

次に、Runを作成し、モデルがFile Searchツールを使用してユーザーの質問に対するレスポンスを提供することを確認します。

```

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 | ストリーミング

アシスタントからの完全なレスポンスを待つのではなく、Server-Sent Events を使用してレスポンスをストリーミングできます。**CreateRun** 関数を使用する際に **Stream = True** パラメータを渡すだけで、レスポンスはストリーミングを使用します。

ストリーミングレスポンスを処理するために次のイベントが使用されます:

- **OnStreamRun:** runオブジェクトに更新があったときに呼び出されるイベントです。
- **OnStreamMessage:** メッセージオブジェクトに更新がある場合 (作成、処理中、完了など) にこのイベントが呼び出されます。
- **OnStreamMessageDelta:** メッセージの一部がストリーミングされているときに発生します。
- **OnStreamDone:** ストリームが終了したときに発生します。
- **OnStreamError:** エラーが発生したときに起動します。内部サーバーエラーまたはタイムアウトが原因となる場合があります。

ステップ 1: アシスタントを作成する

アシスタントは、モデル、指示、ツールなどのいくつかのパラメータを使用してユーザーのメッセージに返答するよう設定できるエンティティを表します。

```
// 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();
```

ステップ2: スレッドの作成

Thread はユーザーと 1 つまたは複数の Assistant 間の会話を表します。ユーザー (または AI アプリケーション) が Assistant との会話を開始するときに Thread を作成できます。

```
oThread = oAssistant->CreateThread();
```

ステップ 3: スレッドにメッセージを追加してストリーミングで実行する

ユーザーまたはアプリケーションが作成したメッセージの内容は、スレッドにメッセージオブジェクトとして追加されます。メッセージにはテキストとファイルの両方を含めることができます。スレッドに追加できるメッセージ数に制限はありません。モデルのコンテキストウィンドウに収まらないコンテキストはスマートに切り捨てられます。

すべてのユーザーメッセージがスレッドに追加されたら、任意のアシスタントでスレッドを実行できます。実行を作成すると、アシスタントに関連付けられたモデルとツールを使用して応答が生成されます。応答は Server-Sent ストリームを使用して送信されます。

```
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);
    }
}
```

ステップ 4: レスポンスの処理

OnStreamMessageDelta イベントを使用してサーバー送信ストリームメッセージを読み取ります。

```
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 関数呼び出し

Chat Completions API と同様に、Assistants API は関数呼び出しをサポートしています。関数呼び出しを使用すると、関数を Assistants API に記述し、呼び出す必要がある関数とその引数をインテリジェントに返させることができます。

この例では、天気アシスタントを作成し、アシスタントが呼び出せるツールとして **get_current_temperature** と **get_rain_probability** という 2 つの関数を定義します。並列関数呼び出しを使用するこの例では、サンフランシスコの今日の天気と降雨確率をアシスタントに尋ねます。また、ストリーミングを使用してアシスタントのレスポンスを出力する方法も示します。

ステップ1 関数を定義します。

アシスタントを作成する際、アシスタントの tools パラメータの下に関数を定義します。

```
TsgcAIOpenAIAssistant *Assistant = new TsgcAIOpenAIAssistant(nullptr);
Assistant->OpenAIOptions->ApiKey = "sk-askdjfalskdjfl23kjkjasdefasdfj";
Assistant->AssistantOptions->Name = "Delphi Weather Bot";
Assistant->AssistantOptions->Instructions->Text = "You are a weather bot. Use the provided functions to answer qu";
Assistant->AssistantOptions->Model = "gpt-4o";
Assistant->AssistantOptions->Tools->Functions->Enabled = false;
Assistant->AssistantOptions->Tools->Functions->Functions->Text =
    "[{\"type\":\"function\",\"function\":{\"name\":\"get_current_temperature\",\"description\":\"Get the current
Assistant->AssistantOptions->Tools->FileSearch->Enabled = false;
Assistant->AssistantOptions->Tools->CodeInterpreter->Enabled = false;
```

ステップ2: スレッドを作成してメッセージを追加します。

ユーザーが会話を開始したときにスレッドを作成し、ユーザーが質問するにつれてスレッドにメッセージを追加します。

```
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
    }
}
```

ステップ 3: OnFunctionCall イベントを処理する

コンポーネントが関数パラメータ値が必要であることを検出すると、**OnFunctionCall** イベントが呼び出されます。リクエストの詳細を知るには **Request._Function** パラメータを使用し、レスポンスを送信するには **Response.Output** を使用してください。

```
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

OpenAI API を音声コマンドで使用するには、以下の手順が必要です。

1. マイクの音声をキャプチャする必要があるため、テキスト変換のために音声認識システムが必要です。
 1. マイクの音声のキャプチャは [TsgcAudioRecorderMCI](#) コンポーネントを使用して行います。
 2. 音声をキャプチャしたら、OpenAI Whisper API に送信して音声ファイルをテキストに変換します。
2. 音声認識の結果を取得したら、ChatCompletion API を使用してテキストを OpenAI に送信します。
3. OpenAI からのレスポンスは、次のいずれかのコンポーネントを使用して音声に変換する必要があります：
 1. [TsgcTextToSpeechSystem](#): (現在 Windows のみ対応) オペレーティングシステムの Windows テキスト読み上げを使用します。
 2. [TsgcTextToSpeechGoogle](#): OpenAI からのレスポンスを Google Cloud サーバーに送信し、返された MP3 ファイルを [TsgcAudioPlayerMCI](#) で再生します。
 3. [TsgcTextToSpeechAmazon](#): OpenAI からのレスポンスを Amazon AWS サーバーに送信し、返された MP3 ファイルを [TsgcAudioPlayerMCI](#) で再生します。

コンポーネント

以下のコンポーネントは、マイクからの音声キャプチャ、音声ファイルの再生、テキストから音声への変換に使用されます。

- [TsgcAudioRecorderMCI](#): (Windows のみ) このコンポーネントを使用してマイクにアクセスし、音声を wave ファイルに変換できます。
- [TsgcAudioPlayerMCI](#): (Windows のみ) mp3 ファイルを再生するコンポーネント。
- [TsgcTextToSpeechSystem](#): (Windows のみ) 外部 mp3 ファイルを使用せずにテキストを音声に変換します。
- [TsgcTextToSpeechGoogle](#): 利用可能な Google Cloud の音声を使用してテキストを音声に変換します。
- [TsgcTextToSpeechAmazon](#): 利用可能な Amazon AWS の音声を使用してテキストを音声に変換します。

TsgcAudioRecorderMCI

このコンポーネントはマイクのオーディオをキャプチャして WAV ファイルに保存するために使用します。

はじめに

このコンポーネントはマイクの音声をキャプチャしてウェーブファイルに保存するために使用します。現在は Windows のみがサポートされています。

リファレンス

- プロパティ — コンポーネントで利用可能なプロパティ3件。
- メソッド — 2個のパブリックメソッド。
- イベント — 処理可能なイベントは0件です。

TsgcAudioPlayerMCI

このコンポーネントは、*text-to-speech* プロバイダーから受信したmp3ファイルを再生するために使用されます。

はじめに

このコンポーネントは、テキスト読み上げ プロバイダーから受信した mp3 ファイルを再生するために使用されます。現在は Windows のみサポートされています。

リファレンス

- プロパティ — コンポーネントで利用可能な 1 つのプロパティ。
- メソッド — 1 つのパブリックメソッド。
- イベント — 処理可能なイベントは 0 件です。

TsgcTextToSpeechSystem

これはオペレーティングシステムが提供するデフォルトのテキスト読み上げ機能です。現在は Windows のみ対応しています。

はじめに

これはオペレーティングシステムが提供するデフォルトのテキスト読み上げ機能です。現在は Windows のみ対応しています。

リファレンス

- プロパティ — コンポーネントで利用可能な1個のプロパティ。
- メソッド — 1つのパブリックメソッド。
- イベント — 処理可能なイベントは0件です。

TsgcTextToSpeechGoogle

Text-To-Speech は、テキストを MP3 ファイルに変換できる Google Cloud が提供する API です。

はじめに

Text-To-Speech は、テキストをmp3ファイルに変換できる Google Cloud が提供するAPIです。Google Cloud アカウントと、Text-To-Speech アカウントのセットアップが必要です。

Text-To-Speech アカウントが設定されたら、JSON 設定ファイルをダウンロードして `GoogleOptions.Settings` プロパティに設定する必要があります。

リファレンス

- プロパティ — コンポーネントで利用可能な3個のプロパティ。
- メソッド — 1つのパブリックメソッド。
- イベント — 処理できるイベントは0件です。

TsgcTextToSpeechAmazon

Text-To-Speech は Amazon AWS が提供する API で、テキストを MP3 ファイルに変換できます。

はじめに

Text-To-Speech は Amazon AWS が提供する API であり、テキストを mp3 ファイルに変換できます。Amazon AWS アカウントと Polly API の設定が必要です。

リファレンス

- プロパティ — コンポーネントで利用可能な 3 つのプロパティ。
- メソッド — 1 つのパブリックメソッド。
- イベント — 処理できるイベント 0 件。

TsgcAIChat

TsgcAIChat は任意の AI プロバイダーと連携する単一のコンポーネントです。

はじめに

TsgcAIChat は任意の AI プロバイダーと連携する単一のコンポーネントです。プロパティを1つ変更するだけでプロバイダーを切り替えられます。コンポーネントには組み込みの会話履歴管理が含まれており、メッセージを手動で追跡することなくチャットアプリケーションを簡単に構築できます。

サポートされているプロバイダー

- OpenAI
- Anthropic
- Gemini
- DeepSeek
- Ollama
- Grok
- Mistral

リファレンス

- プロパティ — コンポーネントで利用可能な 5 つのプロパティ。
- メソッド — パブリックメソッド 4 個。
- イベント — 処理できる3つのイベント。

TsgcAIOpenAIChatBot

マイクの音声をキャプチャする必要があるため、OpenAIに送信するテキストを取得するには音声認識システムが必要です。

はじめに

音声コマンドを使用したチャットボットを構築するには、以下の手順が必要です:

1. マイクの音声をキャプチャする必要があるため、テキストに変換する音声認識システムが必要です。取得したテキストが OpenAI に送信されます。
 1. マイク音声のキャプチャは [TsgcAudioRecorderMCI](#) コンポーネントを使用して行います。
 2. 音声をキャプチャしたら、OpenAI Whisper API に送信して音声ファイルをテキストに変換します。
2. 音声認識の結果を取得したら、ChatCompletion API を使用してテキストを OpenAI に送信します。
3. OpenAI からの応答はその後、以下のコンポーネントのいずれかを使用して音声に変換する必要があります:
 1. [TsgcTextToSpeechSystem](#): (現在 Windows のみ) オペレーティングシステムの Windows テキスト読み上げ機能を使用します。
 2. [TsgcTextToSpeechGoogle](#): OpenAI からの応答を Google Cloud サーバーに送信し、返された mp3 ファイルを [TsgcAudioPlayerMCI](#) で再生します。
 3. [TsgcTextToSpeechAmazon](#): OpenAI からの応答を Amazon AWS サーバーに送信し、返された mp3 ファイルを [TsgcAudioPlayerMCI](#) で再生します。

リファレンス

- プロパティ — コンポーネントで利用可能な6つのプロパティ。
- メソッド — 4個のpublicメソッド。
- イベント — 処理できる 4 つのイベント。

TsgcAIOpenAITranslator

マイクの音声をキャプチャする必要があるため、OpenAIに送信するテキストを取得するには音声認識システムが必要です。

はじめに

音声コマンド付きのトランスレーターを構築するには、次の手順が必要です。

1. マイクの音声をキャプチャする必要があるため、テキストに変換する音声認識システムが必要です。取得したテキストが OpenAI に送信されます。
 1. マイク音声のキャプチャは [TsgcAudioRecorderMCI](#) コンポーネントを使用して行います。
 2. 音声をキャプチャしたら、OpenAI Whisper API に送信して音声ファイルをテキストに変換します。
2. 音声認識の結果を取得したら、ChatCompletion API を使用してテキストを OpenAI に送信します。
3. OpenAI からの応答はその後、以下のコンポーネントのいずれかを使用して音声に変換する必要があります:
 1. [TsgcTextToSpeechSystem](#): (現在 Windows のみ) オペレーティングシステムの Windows テキスト読み上げ機能を使用します。
 2. [TsgcTextToSpeechGoogle](#): OpenAI からの応答を Google Cloud サーバーに送信し、返された mp3 ファイルを [TsgcAudioPlayerMCI](#) で再生します。
 3. [TsgcTextToSpeechAmazon](#): OpenAI からの応答を Amazon AWS サーバーに送信し、返された mp3 ファイルを [TsgcAudioPlayerMCI](#) で再生します。

リファレンス

- プロパティ — コンポーネントに 5 つのプロパティがあります。
- メソッド — パブリックメソッド2件。
- イベント — 処理可能な3個のイベント。

TsgcAIOpenAIEmbeddings

埋め込みは、単語、フレーズ、または画像や音声などの他の種類のデータを数値形式で表現する方法です。

はじめに

エンベディングは、単語、フレーズ、さらには画像や音声などの他の種類のデータを数値形式で表現する方法です。単語やデータをコンピューターが理解しやすく処理できるように数値に変換するようなものです。

「dog」「cat」「bird」のような単語の集まりがあると想像してください。これらの単語には意味がありますよね。エンベディングは、各単語に意味や他の単語との関係を捉えた一意の数値の集合（ベクトル）を割り当てます。

例えば、「dog」という単語は [0.5, 0.2, -0.7]、「cat」は [0.8, -0.3, 0.1]、「bird」は [0.3, 0.9, 0.4] で表されます。ベクター内の数値は各単語の特性に関する情報を含んでいます。例えば、動物に関連しているかどうか、またはお互いにどれほど似ているかなどです。

素晴らしいことに、エンベディングは大量のデータから学習できるため、単語間の類似点と相違点を自動的に見つけることができます。これらの数値表現により、AI アルゴリズムが言語を理解し、複雑なパターンを把握することが可能になります。これは、言語翻訳、感情分析、推薦システムなどのさまざまなアプリケーションで重要です。また、AI モデルが情報を処理してより正確な結果を提供するのが容易かつ高速になります！

リファレンス

- プロパティ — コンポーネントで利用可能な4つのプロパティ。
- メソッド — 2個のpublicメソッド。
- イベント — 処理できる 5 つのイベント。

TsgcAIDatabaseVectorFile

コンポーネントはデータベースベクターとプロンプトを2つのテキストファイルに保存します。

はじめに

このコンポーネントはデータベースのベクターとプロンプトを2つのテキストファイルに保存します。ベクター数が多い場合は最適化されていないため、本番環境ではなくテスト目的のみで使用してください。

設定

- **VectorFileOptions:**
 - **InputFilename:** 入力データの保存に使用するファイルの名前。
 - **VectorFilename:** ベクターの保存に使用するファイルの名前。

リファレンス

- プロパティ — コンポーネントで利用可能なプロパティが2つあります。
- メソッド — 4個のpublicメソッド。
- イベント — 処理可能なイベントは0件です。

TsgcAIDatabaseVectorPinecone

TsgcAIDatabaseVectorPinecone コンポーネント。

はじめに

このコンポーネントは REST [Pinecone API クライアント](#) をベースにしており、インデックスとベクターの作成・更新・削除が可能です。

設定

- **PineconeOptions:**
 - **ApiKey:** pinecone から提供される API キーをここで設定します。これは pinecone アカウントから取得できます。
 - **Environment:** デフォルトは無料アカウントの "us-west4-gcp-free" です。
 - **LogOptions:** HTTP リクエストをテキストファイルに保存したい場合は、ここで設定します。
- **PineconeIndexOptions:**
 - **IndexName:** データの保存またはクエリに使用されるインデックスの名前です。
 - **ProjectId:** プロジェクトの ID です。

リファレンス

- プロパティ — コンポーネントで利用可能なプロパティ 3 件。
- メソッド — パブリックメソッド 4 個。
- イベント — 処理可能なイベントは 0 件です。

Embeddings | Create Vectors

埋め込みを使用するには、まずデータをベクターに変換する必要があります。

使用例

PDF ファイルがある場合は、まず PDF をテキストに変換し、その後 **CreateEmbeddingsFromFile** メソッドを使用してベクターデータを取得します。OpenAI 埋め込みのサイズ制限により、ファイルが大きすぎる場合はデータが自動的にチャンクに分割されます。1 つのファイルから 1 つまたは複数のベクターを取得できます。

以下にコードサンプルを示します。

```
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;
    }
}
```

エンベディング | ChatBot

すべてのデータをベクターに変換したら、独自のモデルの構築を開始できます。考え方は非常にシンプルです。ボットに質問するたびに、まず質問をベクターに変換し、次にデータベースでその質問に最も類似するベクターを検索し、最後に最も類似するデータをコンテキストとして追加します。

```
void AskToChatGPT(const std::string& aQuestion)
{
    TsgcAIOpenAIChatBot* oChatBot = new TsgcAIOpenAIChatBot(NULL);
    try
    {
        oChatBot->OpenAIOptions->ApiKey = "<your api key>";
        TsgcAIOpenAIEmbeddings* oEmbeddings = new TsgcAIOpenAIEmbeddings(NULL);
        try
        {
            oChatBot->Embeddings = oEmbeddings;
            TsgcAIDatabaseVectorFile* oFile = new TsgcAIDatabaseVectorFile(NULL);
            try
            {
                oEmbeddings->Database = oFile;
                std::string vContext = oChatBot->GetEmbedding(aQuestion);
                std::string message = "Answer the question based on the context below.\n\nContext:\n" +
                    vContext + "\nQuestion:" + aQuestion + "\nAnswer:";
                oChatBot->ChatAsUser(message.c_str());
            }
            __finally
            {
                delete oFile;
            }
        }
        __finally
        {
            delete oEmbeddings;
        }
    }
    __finally
    {
        delete oChatBot;
    }
}
```

Pinecone

[Pinecone.io](https://pinecone.io)

Pinecone は、ベクトルデータを簡単かつ強力な方法でアップロード、クエリ、削除できるベクトルデータベースです。

Pinecone には、サードパーティが Pinecone を独自のアプリケーションに統合できるパブリック API があります。コンポーネント `TsgcHTTP_API_Pinecone` は Pinecone API のラッパーです。

設定

開始前に Pinecone ウェブサイトに登録して API キーを申請する必要があります。この API キーは API リクエストの送信に使用され、`TsgcHTTP_API_Pinecone` コンポーネントの **`PineconeOptions.ApiKey`** プロパティに設定する必要があります。

インデックス操作

以下のメソッドがサポートされています:

メソッド	パラメータ	説明
<code>IndexesList</code>		この操作は、Pinecone インデックスのリストを返します。
<code>IndexCreate</code>	<code>TsgcHTTPIPineconeIndexCreate</code>	この操作は Pinecone インデックスを作成します。類似度の尺度、インデックスに保存するベクターの次元数、使用するレプリカの数などを指定できます。
<code>IndexDescribe</code>	インデックス名	インデックスの説明を取得します。
<code>IndexDelete</code>	インデックス名	この操作は既存のインデックスを削除します。
<code>IndexConfigure</code>	インデックス名、レプリカ数、PodType	この操作はインデックスのポッドタイプとレプリカ数を指定します。

コレクション操作

以下のメソッドがサポートされています:

メソッド	パラメータ	説明
<code>CollectionsList</code>		この操作はあなたの Pinecone コレクションの一覧を返します。
<code>CollectionCreate</code>	コレクション名、ソース	この操作は Pinecone コレクションを作成します。
<code>CollectionDescribe</code>	コレクション名	コレクションの説明を取得します。
<code>CollectionDelete</code>	コレクション名	この操作は既存のコレクションを削除します。

ベクター操作

以下のメソッドがサポートされています:

メソッド	パラメータ	説明
VectorsDescribeIndexStats	Index Name, Project Id, Filter	DescribeIndexStats 操作は、名前空間ごとのベクター数と次元数を含む、インデックスのコンテンツに関する統計を返します。
VectorsQuery	インデックス名、プロジェクト ID、パラメーター	クエリ操作は名前空間をクエリベクターを使用して検索します。名前空間内の最も類似したアイテムの ID とその類似スコアを取得します。
VectorsDelete	インデックス名、プロジェクト ID、パラメーター	Delete 操作は、単一の名前空間から id でベクターを削除します。id によってアイテムを単一の名前空間から削除できます。
VectorsFetch	インデックス名、プロジェクト ID、ID	Fetch 操作は単一のネームスペースから ID でベクトルを検索して返します。返されるベクトルにはベクトルデータおよび/またはメタデータが含まれます。
VectorsUpdate	インデックス名、プロジェクト ID、パラメーター	Update 操作は名前空間内のベクターを更新します。値が含まれている場合は前の値を上書きします。set_metadata が含まれている場合は、指定されたフィールドの値が追加されるか、前の値を上書きします。
VectorsUpsert	インデックス名、プロジェクト ID、パラメーター	Upsert 操作はベクターを名前空間に書き込みます。既存のベクター ID に新しい値がアップサートされると、以前の値が上書きされます。

UPSERT の使用例

以下に、Id = "id1" の単一ベクトルを UPSERT する使用例を示します。

```
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";
        TsgcHTTPPineconeVectorUpserts* oParams = new TsgcHTTPPineconeVectorUpserts();
        try
        {
            TsgcArrayOfVectorUpsert oVectors;
            oVectors.push_back(new TsgcHTTPPineconeVectorUpsert());
            oVectors[0]->Id = "id1";
            oVectors[0]->Values = aVector;
            oParams->Vectors = oVectors;
            oPinecone->VectorsUpsert(aIndexName, aProjectId, oParams);
        }
        __finally
    }
```



```
        {
            oParams->Free();
        }
    }__finally
    {
        oPinecone->Free();
    }
}
```

例: QUERY

単一のベクターをクエリする例を以下に示します。

```
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

Anthropic は信頼性が高く、解釈可能で、制御可能な AI システムを構築する AI セーフティ企業です。主力モデルファミリーは Claude であり、思慮深い対話、コンテンツ作成、複雑な推論、コーディングなどに優れています。sgcWebSockets ライブラリは、Anthropic Claude API とやり取りするための Delphi コンポーネント `TsgcHTTP_API_Anthropic` を提供します。

Anthropic API

Anthropic API は AI 搭載アプリケーションを構築するための Claude モデルへのアクセスを提供します。API はテキスト生成、ビジョン（画像理解）、ツールの使用（関数呼び出し）、拡張思考、ドキュメント/PDF 処理、プロンプトキャッシング、引用、Web 検索、ストリーミング、トークンカウント、メッセージバッチをサポートしています。

機能

- メッセージ
 - [Anthropic メッセージ使用例](#)
- Vision
 - [Anthropic Vision の例](#)
- ツール使用
 - [Anthropic ツール使用の使用例](#)
- モデル
 - [Anthropic モデル使用例](#)
- バッチ
 - [Anthropic バッチの例](#)
- Extended Thinking
 - [Anthropic Extended Thinking の使用例](#)
- ドキュメント
 - [Anthropic ドキュメントの使用例](#)
- プロンプトキャッシング
 - [Anthropic プロンプトキャッシングの使用例](#)
- Citations
 - [Anthropic Citations Examples](#)
- ウェブ検索
 - [Anthropic ウェブ検索の使用例](#)
- 構造化出力
 - [Anthropic 構造化出力の例](#)
- ファイル
 - [Anthropic Files API 使用例](#)
- MCP コネクタ
 - [Anthropic MCP コネクタの使用例](#)

設定

Anthropic API は認証に API キーを使用します。リクエストで使用する API キーを取得するには、Anthropic Console の [API キー](#) ページを参照してください。

API キーは秘密です！他者と共有したり、クライアント側のコードで公開したりしないでください。

この **API キー** はコンポーネントの `AnthropicOptions.ApiKey` プロパティに設定する必要があります。

`AnthropicOptions.AnthropicVersion` プロパティは API バージョンを指定します（デフォルト: 2023-06-01）。

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "YOUR_API_KEY";
```

プロパティ

AnthropicOptions

- **ApiKey**: Anthropic API で認証するための API キー。
- **AnthropicVersion**: APIバージョン文字列（デフォルト: 2023-06-01）。
- **BetaHeaders**: プレリリース API 機能（例: files-api-2025-04-14、mcp-client-2025-11-20）を有効にするオプションのベータ機能ヘッダー。
- **HttpOptions**: HTTP 接続設定。
 - **ReadTimeout**: HTTP レスポンスを読み取るためのタイムアウト（ミリ秒）。デフォルトは 0（タイムアウトなし）。
- **LogOptions**: ログ設定。
 - **Enabled**: True の場合、HTTP リクエストとレスポンスがファイルに記録されます。
 - **FileName**: ログ出力が書き込まれるファイルパス。
- **RetryOptions**: 失敗したAPIリクエストの自動リトライ設定。
 - **Enabled**: True の場合、失敗したリクエストは自動的にリトライされます。
 - **Retries**: リトライの最大試行回数。
 - **Wait**: リトライ試行間の待機時間（ミリ秒単位）。

メッセージ

テキストおよび/または画像コンテンツを含む入力メッセージの構造化リストを送信し、モデルは会話内の次のメッセージを生成します。

- **CreateMessage**: 指定されたモデルとパラメータでメッセージを作成します。
 - **Model**: 使用するモデル（例: claude-sonnet-4-20250514）。
 - **Message**: ユーザーメッセージの内容。
 - **MaxTokens**: 生成するトークンの最大数（必須、デフォルト 4096）。
- **CreateMessageWithSystem**: システムプロンプト付きのメッセージを作成します。
 - **System**: アシスタントの動作を設定するシステムプロンプト。
- **CreateMessageStream**: ストリーミング（SSE）を有効にしてメッセージを作成します。イベントは OnHTTPAPISSE イベントハンドラーを通じて配信されます。
- **CreateMessageJSON**: 指定された JSON スキーマに準拠した構造化 JSON を返すメッセージを作成します。
 - **Schema**: 出力フォーマットを定義する JSON スキーマ文字列。
- **CreateMessageWithThinking**: 拡張思考を有効にしてメッセージを作成します。
 - **BudgetTokens**: 思考のための最大トークンバジェット（最小1024）。
- **CreateDocumentMessage**: base64エンコードされたドキュメント（PDFまたはテキスト）でメッセージを作成します。
 - **DocumentBase64**: base64エンコードされたドキュメントデータ。
 - **MediaType**: MIMEタイプ（例: application/pdf）。

ビジョン

Claudeはメッセージ内のbase64エンコードされたコンテンツブロックとして渡された画像を理解できます。

- **CreateVisionMessage**: テキストプロンプト付きの画像を送信します。
 - **ImageBase64**: base64 エンコードされた画像データ。
 - **MediaType**: MIME タイプ（image/jpeg、image/png、image/gif、image/webp）。
 - **Prompt**: 画像に添付するテキストプロンプト。

ツール使用

Claude は外部システムと連携するためにツール（関数呼び出し）を使用できます。名前、説明、入力スキーマでツールを定義し、Claude はツールを呼び出したいときに tool_use コンテンツブロックを生成します。

モデル

利用可能な Claude モデルの一覧と説明を表示します。

- **GetModels**: 利用可能なすべてのモデルを一覧表示します。
- **GetModel**: 特定のモデルに関する情報を取得します。
 - **ModelId**: 取得するモデルのID。

拡張思考

拡張思考により、Claudeは応答する前に段階的に推論できるようになり、数学、コーディング、分析などの複雑なタスクの品質が向上します。

- **ThinkingType:** 思考を有効にするには「enabled」に設定し、無効にするには「disabled」に設定します。
- **ThinkingBudgetTokens:** 思考用のトークン予算（最小 1024、MaxTokens 未満でなければなりません）。
- **CreateMessageWithAdaptiveThinking:** 適応型思考でメッセージを作成し、Claude が思考の深さを自動的に決定します。

ドキュメント

ClaudeはPDFドキュメントとテキストファイルをコンテンツブロックとして処理できます。base64、テキスト、URLのソースタイプをサポートします。

- **CreateDocumentMessage:** テキストプロンプト付きのドキュメントを送信します。
 - **DocumentBase64:** base64 エンコードされたドキュメントデータ。
 - **MediaType:** MIME タイプ（application/pdf、text/plain）。

Prompt Caching

頻繁に使用されるコンテキスト(システムプロンプト、コンテンツブロック、ツール定義)をAPI呼び出し間でキャッシュして、キャッシュ読み取り時のコストを最大90%削減します。

- **SystemCacheControl:** システムプロンプトをキャッシュするために、要求でTrueに設定します。
- **CacheControl:** コンテンツブロックまたはツール定義で「ephemeral」に設定します。
- **BetaHeaders:** AnthropicOptions のオプションのベータ機能ヘッダー。

引用

引用が有効な状態でドキュメントが送信されると、Claude はレスポンスにソース参照を含めます。引用タイプには、char_location（テキスト）、page_location（PDF）、content_block_location（カスタムコンテンツ）、web_search_result_location（Web検索）があります。

Web 検索

Claude は組み込みの web_search ツールを使用してリアルタイム情報を Web 検索できます。他の組み込みツールには code_execution と computer use があります。

- **CreateMessageWithWebSearch:** Webサーチを有効にしてメッセージを作成します。
 - **Model:** 使用するモデル。
 - **Message:** ユーザーのクエリ。

トークンカウント

送信する前にメッセージのトークン数をカウントします。

- **CountTokens:** メッセージの入力トークン数をカウントします。
 - **Model:** トークンカウントに使用するモデル。
 - **Message:** トークンをカウントするメッセージの内容。

メッセージバッチ

Message Batches API を使用すると、大量のメッセージを非同期で処理できます。

- **ListBatches:** すべてのメッセージバッチを一覧表示します。
- **GetBatch:** ID で特定のバッチを取得します。
- **CancelBatch:** まだ処理中のバッチをキャンセルします。
- **GetBatchResults:** 完了したバッチの結果を取得します。

構造化出力

提供された JSON スキーマに準拠した有効な JSON を返すよう Claude に強制します。Effort パラメータと組み合わせで出力品質とコストのバランスを制御してください。

- **CreateMessageJSON:** JSON スキーマ出力でメッセージを作成します。
 - **Schema:** 出力形式を定義する JSON スキーマ文字列。
- **OutputFormatSchema:** （リクエストプロパティ）構造化出力の JSON スキーマ。
- **Effort:** （Requestプロパティ）品質とコストを制御します: 'low'、'medium'、'high'、'max'。
- **Strict:** （ツールプロパティ）True の場合、ツールの入力が入力_schema に一致することが保証されます。

Files API

ファイルをアップロード、一覧表示、取得、ダウンロード、削除します。アップロードされたファイルは、ファイルソースタイプのドキュメントコンテンツブロックを使用してメッセージ内で参照できます。ベータヘッダー `files-api-2025-04-14` が必要です。

- **UploadFile:** ローカルファイルをアップロードします。ファイルメタデータを返します。
- **ListFiles:** ページネーション付きでアップロードされたファイルを一覧表示します。
- **GetFile:** 特定のファイルのメタデータを取得します。
- **DownloadFile:** ファイルコンテンツをダウンロードします。
- **DeleteFile:** ファイルを完全に削除します。

リクエストパラメーター

Messages APIで利用可能な追加のリクエストパラメーター。

- **ServiceTier:** 優先ティアの使用を制御します: 'auto' (デフォルト) または 'standard_only'。
- **InferenceGeo:** 推論の地域を制御します。'global' (デフォルト) または 'us'。
- **Container:** リクエスト間でコード実行環境を再利用するためのコンテナ ID。
- **IsError:** (コンテンツブロックプロパティ) ツールの実行失敗を示すために `tool_result` ブロックで `True` に設定します。
- **CacheTTL:** 拡張キャッシュ存続時間: '5m' (デフォルト) または '1h'。
- **ThinkingType:** Claude が思考の深さを自動的に決定するには「adaptive」に設定します。

MCP コネクタ

Claude をサードパーティツールにアクセスするための外部 MCP (Model Context Protocol) サーバーに接続します。ベータヘッダー `mcp-client-2025-11-20` が必要です。

- **CreateMessageWithMCP:** MCP サーバー接続でメッセージを作成します。
 - **MCPServerUrl:** MCP サーバーの HTTPS URL。
 - **MCPServerName:** サーバーの一意の名前。
- **MCPServers:** (リクエストプロパティ) `ServerType`、`Url`、`Name`、`AuthorizationToken` を持つ MCP サーバー定義の配列。
- **MCPServerName:** (Tool プロパティ) `ToolType` が 'mcp_toolset' の場合に MCP サーバーを参照します。

TsgcHTTP_API_Anthropic

Claude のメッセージ、ファイル、バッチ、トークンカウント用の Anthropic REST API クライアント。

はじめに

Claude のメッセージ、ファイル、バッチ、トークンカウント用の Anthropic REST API クライアント。

はじめに

Claude のメッセージ、ファイル、バッチ、トークンカウント用の Anthropic REST API クライアント。
API キーとオプションの HTTP/リトライ/ログ設定で **AnthropicOptions** プロパティを設定し、リクエストオブジェクトを使用して CreateMessage/CreateChatCompletion などのタイプ付きメソッドを呼び出して、タイプ付きレスポンスクラスを受け取ります。
プロバイダーの背景については、[Anthropic](#) ガイドを参照してください。

リファレンス

- プロパティ — コンポーネントで使用可能な5つのプロパティ。
- メソッド — パブリックメソッド 7 個。
- イベント — 処理可能な 2 つのイベント。

Anthropic | Messages

テキストコンテンツを含む入力メッセージの構造化されたリストを送信すると、モデルは会話の次のメッセージを生成します。

簡単な例

Claude にハローメッセージを送信します。

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->CreateMessage("claude-sonnet-4-20250514", "Hello!"));
```

システムプロンプトの例

Claudeの動作を制御するシステムプロンプト付きのメッセージを送信します。

```
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?"));
```

高度な例

メッセージパラメータを完全に制御するには、型付きリクエスト/レスポンスクラスを使用してください。

```
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;
}
```

ストリーミングの使用例

Server-Sent Events (SSE) を使用して応答をリアルタイムでストリーミングします。OnHTTPAPISSEイベントハンドラを割り当てて、ストリーミングイベントを受信してください。

```
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 | ビジョン

Claude は画像を理解して分析できます。コンテンツブロック内でbase64エンコードされたデータとして画像を送信できます。

サポートされている画像形式

- **image/jpeg** - JPEG画像
- **image/png** - PNG 画像
- **image/gif** - GIF 画像
- **image/webp** - WebP 画像

簡単な例

Claude に画像とともにプロンプトを送信してその画像を説明させます。

```
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"));
```

高度な例

画像メッセージをより細かく制御するためにコンテンツブロックを使用します。

```
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

Claude はツールの使用（関数呼び出し）をサポートしており、会話中に Claude が呼び出せるツールを定義できます。Claude がツールを使用することを決定すると、**tool_use** コンテンツブロックを返します。その後ツールを実行して、結果を **tool_result** コンテンツブロックとして送り返します。

ツール使用フロー

1. 名前、説明、input_schema（JSON スキーマ）を持つツールを定義します。
2. ツールを定義したメッセージを送信します。
3. Claude は **tool_use** コンテンツブロックで返答します（stop_reason = 'tool_use'）。
4. 提供された入力でツールを実行します。
5. **tool_result** コンテンツブロックに出力を含む新しいメッセージを送信します。
6. Claude が最終的な回答を返します。

使用例

天気ツールを定義し、ツール使用ループを処理します。

```
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\" \n    },\n    \"required\": [\"location\"] \n  }\n}";
    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 | モデル

利用可能な Claude モデルを一覧表示して情報を取得します。

モデルの一覧

利用可能なすべての Claude モデルを一覧表示します。

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_GetModels());
```

モデルを取得する

特定のモデルに関する情報を取得します。

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_GetModel("claude-sonnet-4-20250514"));
```

型付きレスポンス

モデルデータへの構造化アクセスには、型付きレスポンスクラスを使用してください。

```
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

メッセージバッチAPIを使用すると、大量のメッセージを非同期で処理できます。これは、即時のレスポンスを必要としないタスク（バルクコンテンツ生成、データ分析、バッチ処理ワークフローなど）に最適です。

バッチの一覧表示

組織のすべてのメッセージバッチを一覧表示します。

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_ListBatches());
```

バッチの取得

特定のバッチに関する情報を取得します。

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_GetBatch("batch_id_here"));
```

バッチのキャンセル

まだ処理中のバッチをキャンセルします。

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_CancelBatch("batch_id_here"));
```

バッチ結果を取得する

完了したバッチの結果を取得します。

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
ShowMessage(Anthropic->_GetBatchResults("batch_id_here"));
```

型付きレスポンス

バッチデータへの構造化されたアクセスには、型付きレスポンスクラスを使用してください。

```
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

拡張思考により、Claudeは応答を提供する前に複雑な問題を段階的に考え抜く能力を得ます。有効にすると、Claudeは内部の推論（思考ブロック）を作成し、数学、コーディング、分析、その他の複雑なタスクに対する応答の品質を向上させます。

簡単な例

コンビニエンスメソッドを使用して、拡張思考を有効にしたメッセージを送信します。思考が有効な場合、API が要求するため、温度は自動的に1.0に設定されます。

```
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));
```

高度な例

型付きリクエスト/レスポンスクラスを使用して完全に制御してください。ThinkingType を 'enabled' に設定し、ThinkingBudgetTokens を目的のトークン予算（最小 1024）に設定します。レスポンスには thinking とテキストコンテンツブロックが含まれます。

```
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;
}
```

思考付きのマルチターン

マルチターン会話で拡張思考を使用する場合、以前のレスポンスからの thinking および redacted_thinking ブロックを会話に含めて渡す必要があります。ContentBlocks 配列を使用してこれらのブロックを含めてください。

```
// 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;
```

プロパティ

- **ThinkingType:** 拡張思考を有効にするには 'enabled' を、無効にするには 'disabled' を設定します。
- **ThinkingBudgetTokens:** 思考に使用するトークンの最大数（最小 1024）。MaxTokens より小さい値である必要があります。

注意

- thinking が有効な場合、Temperature は 1.0 でなければなりません (便利メソッドは自動的に設定します)。
- thinkingが有効な場合、TopKは使用できません。
- BudgetTokens は少なくとも 1024 以上で MaxTokens 未満である必要があります。
- 応答コンテンツブロックには、thinking、redacted_thinking、text タイプが含まれる場合があります。

Anthropic | ドキュメント

Claude はメッセージ内のコンテンツブロックとして送信された PDF ドキュメントとテキストファイルを処理できます。ドキュメントは base64 エンコードされたデータまたは URL で送信され、Claude はそのコンテンツを分析、要約、および質問に回答できます。

簡単な例

便利なメソッドを使用して質問付きの PDF ドキュメントを送信します。

```
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"));
```

引用付きの高度な使用例

引用を有効にしてドキュメントを送信するには、型指定されたクラスを使用します。引用が有効な場合、Claude のレスポンスにはソースドキュメントの特定部分への参照が含まれます。

```
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;
}
```

プロパティ

- **ContentType:** ドキュメントコンテンツブロックには 'document' に設定します。
- **SourceType:** ソースタイプ: 'base64' (デフォルト)、'text'、または 'url'。
- **MediaType:** MIME タイプ (例: application/pdf、text/plain)。
- **Data:** base64 エンコードされたドキュメントデータ、または SourceType が 'url' の場合は URL。
- **Title:** 参照用のオプションのドキュメントタイトル。
- **Context:** ドキュメントに関するオプションのコンテキストまたはメタデータ。
- **CitationsEnabled:** True に設定してレスポンスでのソース引用を有効にします。

サポートされている形式

- **PDF:** application/pdf (最大 32 MB、リクエストあたり最大 100 ページ)
- **プレーンテキスト:** text/plain

Anthropic | プロンプトキャッシング

プロンプトキャッシングにより、API 呼び出し間で頻繁に使用するコンテキストをキャッシュし、キャッシュ読み取り時のコストを最大 90% 削減してレイテンシーを改善できます。キャッシュされたコンテンツは `cache_control` パラメータでマークされ、キャッシュ TTL ウィンドウ内のリクエスト間で再利用されます。

システムプロンプトキャッシング

すべてのリクエストで再処理されないように長いシステムプロンプトをキャッシュします。SystemCacheControl を True に設定すると、システムプロンプトを `cache_control` で自動的にラップします。

```
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();
}
```

コンテンツブロックキャッシング

個々のコンテンツブロックで CacheControl を 'ephemeral' に設定することで、特定のコンテンツブロック（大きなドキュメントやコンテキストなど）をキャッシュします。

```
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
```

ツール定義のキャッシング

すべてのリクエストで再処理されないようにツール定義をキャッシュします。これは、リクエスト間で一定のままである多数のツール定義がある場合に役立ちます。

```
TsgcAnthropicClass_Request_Tool *oTool = new TsgcAnthropicClass_Request_Tool(NULL);
oTool->Name = "search_database";
oTool->Description = "Search the database for records.";
oTool->InputSchema = "{\"type\":\"object\",\"properties\":{\"query\":{\"type\":\"string\"}}\"";
oTool->CacheControl = "ephemeral"; // Cache this tool definition
```

プロパティ

- **SystemCacheControl** : リクエストでシステムプロンプトを一時的な `cache_control` でキャッシュするには `True` に設定します。
- **CacheControl**: コンテンツブロックまたはツール定義で 'ephemeral' に設定すると、キャッシュ対象としてマークされます。
- **CacheCreationInputTokens**: (レスポンス) キャッシュエントリの作成に使用されたトークン数。
- **CacheReadInputTokens**: (レスポンス) キャッシュから読み取ったトークン数 (コスト削減)。

価格設定

- **キャッシュ書き込み**: 基本入力トークン価格の 1.25 倍 (TTL 5 分)。
- **キャッシュ読み取り**: 基本入力トークン価格の 0.1 倍 (90% 節約)。
- **Cache TTL**: デフォルトでは 5 分。後続のリクエストは TTL を更新します。

Anthropic | 引用

引用を有効にしてドキュメントを送信すると、Claudeのレスポンスにはソースドキュメントの特定の部分への参照が含まれます。これにより、主張を検証し、情報の出所を追跡できます。

引用の有効化

ドキュメントコンテンツブロックで `CitationsEnabled` を `True` に設定します。引用はリクエスト内のすべてのドキュメントで有効にするか、すべてで無効にする必要があります。

```
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
```

レスポンスからの引用の読み取り

レスポンスのテキストコンテンツブロックにはソース参照を含む `Citations` 配列が含まれている場合があります。

```
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();
}
```

引用タイプ

- **char_location:** プレーンテキストドキュメントにおける文字レベルの参照。フィールド: `DocumentIndex`、`DocumentTitle`、`StartCharIndex`、`EndCharIndex`。
- **page_location:** PDF ドキュメントのページレベル参照。フィールド: `DocumentIndex`、`DocumentTitle`、`StartPageNumber`、`EndPageNumber`。
- **content_block_location:** カスタムコンテンツドキュメントのブロックレベル参照。フィールド: `DocumentIndex`、`DocumentTitle`、`StartBlockIndex`、`EndBlockIndex`。
- **web_search_result_location:** Web 検索結果への参照。フィールド: `Url`、`Title`、`EncryptedIndex`。

Anthropic | Web Search

Web 検索ツールを使用すると、Claude は会話中にリアルタイムの情報をウェブで検索できます。これは Anthropic がホストし自動的に実行する組み込みのサーバーサイドツールです。

簡単な例

ウェブ検索を有効にしたメッセージを作成するための便利なメソッドを使用します。

```
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?"));
```

高度な例

MaxUses などの Web 検索ツールパラメーターを完全に制御するには、型付きクラスを使用してください。

```
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;
}
```

組み込みツールの種類

- **web_search_20250305:** サーバーサイドのWeb検索。プロパティ: Name ('web_search')、MaxUses。
- **code_execution_20250825:** サンドボックス化されたコード実行。プロパティ: Name ('code_execution')、MaxUses。
- **computer_20250124:** コンピューター操作（マウス、キーボード、スクリーンショット）。プロパティ: Name ('computer')、DisplayWidthPx、DisplayHeightPx。

レスポンスコンテンツタイプ

組み込みツールが使用される場合、レスポンスには追加のコンテンツブロックタイプが含まれる場合があります。

- **server_tool_use** : Claude が組み込みツールを呼び出したことを示します。フィールド: Id、Name、Input。
- **web_search_tool_result**: Web 検索結果を含みます。フィールド: ToolUseId、Data (生の JSON コンテンツ)。

Anthropic | 構造化出力

構造化出力は Claude に JSON スキーマに準拠したレスポンスを返すよう強制します。これにより、スキーマ定義に一致する有効で解析可能な JSON 出力が保証されます。

簡単な例

JSONスキーマ出力でメッセージを作成するには、この便利なメソッドを使用します。

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";

String vSchema = "{\n  \"type\": \"object\",\n  \"properties\": {\n    \"name\": {\n      \"type\": \"string\",\n      \"age\": {\n        \"type\": \"integer\"\n      },\n      \"required\": [\"name\", \"age\"],\n      \"additionalProperties\": false\n    }\n  }\n}";

ShowMessage(Anthropic->CreateMessageJSON("claude-sonnet-4-20250514",
  "Extract the name and age: John is 30 years old.", vSchema));
```

高度な例

構造化された出力をeffortパラメータと組み合わせるには、型付きクラスを使用します。

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";

TsgcAnthropicClass_Request_Messages *oRequest = new TsgcAnthropicClass_Request_Messages();
try {
  oRequest->Model = "claude-sonnet-4-20250514";
  oRequest->MaxTokens = 4096;

  // JSON schema for structured output
  oRequest->OutputFormatSchema =
    "{\n  \"type\": \"object\",\n  \"properties\": {\n    \"sentiment\": {\n      \"type\": \"string\",\n      \"enum\": [\"positive\", \"negative\", \"neutral\"]\n    },\n    \"confidence\": {\n      \"type\": \"number\"\n    },\n    \"required\": [\"sentiment\", \"confidence\"],\n    \"additionalProperties\": false\n  }\n}";

  // 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;
}
```

厳格なツール使用

入力が入力スキーマに厳密に準拠するようにツール定義のストリクトモードを有効にします。

```
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}\"};
```

プロパティ

- **OutputFormatSchema:** JSON スキーマ文字列。Claude のレスポンステキストはこのスキーマに準拠した有効な JSON となります。
- **Effort:** 出力品質とコストのバランスを制御します。値: 'low'、'medium'、'high'（デフォルト）、'max'（Opus 4.6 のみ）。
- **Strict:** （ツール上）Trueの場合、ツールの入力 input_schema に準拠することが保証されます。

Anthropic | Files API

Files API を使用すると、ファイルのアップロード、一覧表示、取得、ダウンロード、削除ができます。アップロードされたファイルは、ファイルソースタイプのドキュメントコンテンツブロックを使用してメッセージ内で参照できます。

注意: Files API にはベータヘッダー `files-api-2025-04-14` が必要です。**`AnthropicOptions.BetaHeaders`** プロパティで設定してください。

ファイルのアップロード

```
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;
}
```

ファイルの一覧表示

```
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;
}
```

メッセージでファイルを使用する

SourceType を 'file' に設定したドキュメントコンテンツブロックを使用して、アップロードされたファイルを参照します。

```
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:** ローカルパスからファイルをアップロードします。ファイルのメタデータ (Id、Filename、MimeType、SizeBytes) を返します。

- **ListFiles** : ページネーションサポート付きでアップロードされたすべてのファイルを一覧表示します (Limit、AfterId、BeforeId)。
- **GetFile** : ID で特定のファイルのメタデータを取得します。
- **DownloadFile**: ファイルのコンテンツをダウンロードします (コード実行ツールによって作成されたファイルのみダウンロード可能)。
- **DeleteFile**: ID によってファイルを完全に削除します。

Anthropic | MCP コネクター

MCP (Model Context Protocol) コネクターを使用すると、Claude が外部 MCP サーバーのツールにアクセスできます。これにより、サードパーティのサービスやカスタムツールプロバイダーとの統合が可能になります。

注意: MCP コネクターにはベータヘッダー `mcp-client-2025-11-20` が必要です。これを `AnthropicOptions.BetaHeaders` プロパティに設定してください。

簡単な例

MCP サーバーでメッセージを作成するための便利なメソッドを使用します。

```
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"));
```

高度な例

認証を含む MCP サーバー設定を完全に制御するための型付きクラスを使用します。

```
TsgcHTTP_API_Anthropic *Anthropic = new TsgcHTTP_API_Anthropic(NULL);
Anthropic->AnthropicOptions->ApiKey = "API_KEY";
Anthropic->AnthropicOptions->BetaHeaders = "mcp-client-2025-11-20";

TsgcAnthropicClass_Request_Messages *oRequest = new TsgcAnthropicClass_Request_Messages();
try {
    oRequest->Model = "claude-sonnet-4-20250514";
    oRequest->MaxTokens = 4096;

    // Configure MCP server
    TsgcAnthropicClass_Request_MCPServer *oServer = new TsgcAnthropicClass_Request_MCPServer();
    oServer->ServerType = "url";
    oServer->Url = "https://my-mcp-server.example.com/sse";
    oServer->Name = "my-server";
    oServer->AuthorizationToken = "OAUTH_TOKEN"; // Optional auth
    oServers = oRequest->MCPServers;
    SetLength(oServers, 1);
    oServers[0] = oServer;
    oRequest->MCPServers = oServers;

    // Add MCP toolset
    TsgcAnthropicClass_Request_Tool *oTool = new TsgcAnthropicClass_Request_Tool();
    oTool->ToolType = "mcp_toolset";
    oTool->MCPServerName = "my-server";
    oTools = oRequest->Tools;
    SetLength(oTools, 1);
    oTools[0] = oTool;
    oRequest->Tools = oTools;

    // Add user message
    TsgcAnthropicClass_Request_Message *oMessage = new TsgcAnthropicClass_Request_Message();
    oMessage->Role = "user";
    oMessage->Content = "Search for recent news about AI.";
    oMessages = oRequest->Messages;
    SetLength(oMessages, 1);
    oMessages[0] = oMessage;
    oRequest->Messages = oMessages;

    TsgcAnthropicClass_Response_Messages *oResponse = Anthropic->CreateMessage(oRequest);
    try {
        for (int i = 0; i < oResponse->Content.Length; i++) {
            if (oResponse->Content[i]->ContentType == "text")
                ShowMessage(oResponse->Content[i]->Text);
            else if (oResponse->Content[i]->ContentType == "mcp_tool_use")
                ShowMessage("MCP tool call: " + oResponse->Content[i]->Name +
                    " on " + oResponse->Content[i]->ServerName);
        }
    }
} finally {
```

```
        delete oResponse;
    }
} __finally {
    delete oServer;
    delete oTool;
    delete oMessage;
    delete oRequest;
}
```

MCP サーバプロパティ

- **ServerType** : 現在は 'url' のみサポートされています。
- **Url**: MCPサーバーのHTTPS URL。
- **Name**: このサーバーの一意の識別子。ツールセット内の MCPServerName と一致する必要があります。
- **AuthorizationToken**: 認証済みサーバー用のオプションの OAuth Bearer トークン。

レスポンスコンテンツタイプ

- **mcp_tool_use**: Claude がMCPツールを呼び出しました。フィールド: Id, Name, ServerName, Input。
- **mcp_tool_result**: MCPツール実行の結果。フィールド: ToolUseId、IsError、Data（コンテンツ）。

Gemini

Google Gemini は Google DeepMind が開発したマルチモーダル AI モデルファミリーです。Gemini モデルはテキスト生成、ビジョン、構造化出力、埋め込み、ツール使用をサポートし、AI を活用したアプリケーション構築のための強力な機能を提供します。

sgcWebSockets ライブラリは、Gemini API と連携するための Delphi コンポーネント **TsgcHTTP_API_Gemini** を提供します。

Gemini API

Gemini API は、AI 搭載アプリケーションを構築するための Google Gemini モデルへのアクセスを提供します。この API は、コンテンツ生成、ビジョン（画像理解）、構造化 JSON 出力、ストリーミング、トークンカウント、埋め込み、ツール使用（関数呼び出し）、モデルリストをサポートします。

機能

- メッセージ
 - [Gemini メッセージの例](#)
- Vision
 - [Gemini Vision 使用例](#)
- モデル
 - [Gemini モデルの使用例](#)
- 構造化出力
 - [Gemini 構造化出力例](#)
- トークンカウント
 - [Gemini トークンカウントの使用例](#)
- 埋め込み
 - [Gemini 埋め込み例](#)
- ツール使用
 - [Gemini ツール使用の例](#)

設定

Gemini API は認証に API キーを使用します。リクエストで使用する API キーを取得するには、Google AI Studio の [API キー](#) ページにアクセスしてください。

API キーは秘密です！他者と共有したり、クライアント側のコードで公開したりしないでください。この **API キー** はコンポーネントの **GeminiOptions.ApiKey** プロパティに設定する必要があります。

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "YOUR_API_KEY";
```

メッセージ

Gemini モデルにコンテンツを送信し、生成された応答を受信します。モデルは提供された入力に基づいて次のメッセージを生成します。

- **_CreateContent**：指定されたモデルとユーザープロンプトでコンテンツを作成します。
 - **Model**：使用するモデル（例：gemini-2.0-flash）。
 - **Message**：ユーザーメッセージの内容。
 - **MaxOutputTokens**：生成するトークンの最大数。
- **_CreateContentWithSystem**：システム指示を含むコンテンツを作成します。
 - **Model**：使用するモデル。
 - **System**：モデルの動作を設定するシステム指示。
 - **Message**：ユーザーメッセージのコンテンツ。
 - **MaxOutputTokens**：生成するトークンの最大数。
- **_CreateContentStream**：ストリーミング（SSE）を有効にしてコンテンツを作成します。イベントは OnHTTPAPISSE イベントハンドラーを通じて配信されます。

ビジョン

Gemini モデルはテキストプロンプトとともに base64 エンコードされたコンテンツとして渡された画像を理解できます。

- **_CreateVisionContent:** テキストプロンプト付きの画像を送信します。
 - **Model:** 使用するモデル。
 - **Prompt:** 画像に伴うテキストプロンプト。
 - **Base64:** base64 エンコードされた画像データ。
 - **MediaType:** MIME タイプ (image/jpeg、image/png、image/gif、image/webp)。
 - **MaxOutputTokens:** 生成するトークンの最大数。

構造化出力

JSON スキーマを提供して期待されるレスポンス形式を定義することにより、Gemini モデルから構造化された JSON 出力を生成します。

- **_CreateContentJSON:** 構造化された JSON 出力でコンテンツを作成します。
 - **Model:** 使用するモデル。
 - **Message:** ユーザーメッセージのコンテンツ。
 - **Schema:** 期待される出力構造を定義する JSON スキーマ。
 - **MaxOutputTokens:** 生成するトークンの最大数。

モデル

利用可能な Gemini モデルの一覧を取得して詳細を確認します。

- **_GetModels:** 利用可能なすべてのモデルを一覧表示します。
- **_GetModel:** 特定のモデルの詳細を取得します。
 - **ModelId:** 取得するモデルの識別子です。

トークンカウント

モデルに送信する前にメッセージのトークン数を計算します。

- **_CountTokens:** メッセージのトークンをカウントします。
 - **Model:** トークン化に使用するモデル。
 - **Message:** トークンをカウントするテキストコンテンツ。

エンベディング

Gemini モデルを使用してテキストコンテンツのベクトル埋め込みを生成します。

- **_EmbedContent:** テキストのエンベディングを生成します。
 - **Model:** エンベディング生成に使用するモデル。
 - **Text:** エンベディングを生成するテキストコンテンツ。

TsgcHTTP_API_Gemini

コンテンツ生成、トークンカウント、エンベディング用の Google Gemini REST API クライアント。

はじめに

コンテンツ生成、トークンカウント、エンベディング用の Google Gemini REST API クライアント。

はじめに

コンテンツ生成、トークンカウント、エンベディング用の Google Gemini REST API クライアント。
API キーとオプションの HTTP/リトライ/ログ設定を使用して **GeminiOptions** プロパティを設定し、リクエストオブジェクトとともに `CreateMessage/CreateChatCompletion` などの型付きメソッドを呼び出し、型付きレスポンスクラスを受け取ります。
プロバイダーの背景については、[Gemini](#) ガイドをご参照ください。

リファレンス

- プロパティ — コンポーネントで利用可能な5つのプロパティ。
- メソッド — パブリックメソッド3件。
- イベント — 処理可能なイベントが 2 つあります。

DeepSeek

DeepSeek は、強力なオープンソース言語モデルの開発に特化した中国の AI 企業です。同社のモデルはコーディング、推論、汎用タスクに優れており、競争力のある価格で優れたパフォーマンスを提供します。sgcWebSocketsライブラリは、DeepSeek APIと対話するためのDelphiコンポーネントTsgcHTTP_API_DeepSeekを提供します。

DeepSeek API

DeepSeek APIは、AIを活用したアプリケーションを構築するためのDeepSeekモデルへのアクセスを提供します。APIは、テキスト生成、vision(画像理解)、ストリーミング、モデル一覧をサポートします。APIはOpenAI互換の形式に従っているため、統合が容易です。

機能

- メッセージ
 - [DeepSeek メッセージの使用例](#)
- ビジョン
 - [DeepSeek ビジョンの使用例](#)
- モデル
 - [DeepSeek モデルの使用例](#)

設定

DeepSeek API は認証に API キーを使用します。リクエストで使用する API キーを取得するには、DeepSeek プラットフォームの [API キー](#) ページにアクセスしてください。
APIキーは秘密です！他者と共有したり、クライアント側のコードで公開したりしないでください。
この **APIキー** はコンポーネントの **DeepSeekOptions.ApiKey** プロパティに設定する必要があります。

```
TsgcHTTP_API_DeepSeek *DeepSeek = new TsgcHTTP_API_DeepSeek(NULL);  
DeepSeek->DeepSeekOptions->ApiKey = "YOUR_API_KEY";
```

メッセージ

テキストコンテンツを含む入力メッセージの構造化されたリストを送信すると、モデルは会話の次のメッセージを生成します。

- **_CreateMessage:** 指定されたモデルとユーザープロンプトでメッセージを作成します。
 - **Model:** 使用するモデル（例: deepseek-chat）。
 - **Message:** ユーザーメッセージの内容。
- **_CreateMessageWithSystem:** システムプロンプト付きのメッセージを作成します。
 - **System:** アシスタントの動作を設定するシステムプロンプト。
- **_CreateMessageStream:** ストリーミング（SSE）を有効にしてメッセージを作成します。イベントは OnHTTPAPISSE イベントハンドラーを通じて配信されます。

ビジョン

DeepSeekはメッセージ内のbase64エンコードされたコンテンツとして渡された画像を理解できます。

- **_CreateVisionMessage:** テキストプロンプトと共に画像を送信します。
 - **ImageBase64:** base64エンコードされた画像データ。
 - **MediaType:** MIMEタイプ（image/jpeg、image/png、image/gif、image/webp）。
 - **Prompt:** 画像に添付するテキストプロンプト。

モデル

利用可能な DeepSeek モデルを一覧表示します。

- **_GetModels:** 利用可能なすべてのモデルを一覧表示します。

TsgcHTTP_API_DeepSeek

チャット補完用の DeepSeek REST API クライアント。

はじめに

チャット補完用の DeepSeek REST API クライアント。

はじめに

チャット補完用の DeepSeek REST API クライアント。

DeepSeekOptions プロパティに API キーとオプションの HTTP/リトライ/ログ設定を設定し、リクエストオブジェクトとともに `CreateMessage/CreateChatCompletion` などの型付きメソッドを呼び出して、型付きレスポンスクラスを受け取ります。

プロバイダーの背景については [DeepSeek](#) ガイドを参照してください。

リファレンス

- プロパティ — コンポーネントに 5 つのプロパティがあります。
- メソッド — 1 つのパブリックメソッド。
- イベント — 処理可能な 2 つのイベント。

DeepSeek | メッセージ

テキストコンテンツを含む入力メッセージの構造化されたリストを送信すると、モデルは会話の次のメッセージを生成します。

簡単な例

DeepSeek に Hello メッセージを送信します。

```
TsgcHTTP_API_DeepSeek *DeepSeek = new TsgcHTTP_API_DeepSeek(NULL);
DeepSeek->DeepSeekOptions->ApiKey = "API_KEY";
ShowMessage(DeepSeek->_CreateMessage("deepseek-chat", "Hello!"));
```

システムプロンプトの例

DeepSeek の動作を制御するシステムプロンプトを含むメッセージを送信します。

```
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?"));
```

ストリーミングの使用例

Server-Sent Events (SSE) を使用して応答をリアルタイムでストリーミングします。OnHTTPAPISSE イベントハンドラを割り当てて、ストリーミングイベントを受信してください。

```
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 | ビジョン

DeepSeek は画像を理解・分析できます。コンテンツブロック内に base64 エンコードデータとして画像を送信できます。

Vision Example

プロンプトとともに画像を送信し、DeepSeekにそれを記述するよう求めます。

```
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

利用可能な DeepSeek モデルを一覧表示します。

モデルの一覧

利用可能なすべての DeepSeek モデルを一覧表示します。

```
TsgcHTTP_API_DeepSeek *DeepSeek = new TsgcHTTP_API_DeepSeek(NULL);  
DeepSeek->DeepSeekOptions->ApiKey = "API_KEY";  
ShowMessage(DeepSeek->_GetModels());
```

Gemini

Google Gemini は Google DeepMind が開発したマルチモーダル AI モデルファミリーです。Gemini モデルはテキスト生成、ビジョン、構造化出力、埋め込み、ツール使用をサポートし、AI を活用したアプリケーション構築のための強力な機能を提供します。

sgcWebSockets ライブラリは、Gemini API と連携するための Delphi コンポーネント **TsgcHTTP_API_Gemini** を提供します。

Gemini API

Gemini API は、AI 搭載アプリケーションを構築するための Google Gemini モデルへのアクセスを提供します。この API は、コンテンツ生成、ビジョン（画像理解）、構造化 JSON 出力、ストリーミング、トークンカウント、埋め込み、ツール使用（関数呼び出し）、モデルリストをサポートします。

機能

- メッセージ
 - [Gemini メッセージの例](#)
- Vision
 - [Gemini Vision 使用例](#)
- モデル
 - [Gemini モデルの使用例](#)
- 構造化出力
 - [Gemini 構造化出力例](#)
- トークンカウント
 - [Gemini トークンカウントの使用例](#)
- 埋め込み
 - [Gemini 埋め込み例](#)
- ツール使用
 - [Gemini ツール使用の例](#)

設定

Gemini API は認証に API キーを使用します。リクエストで使用する API キーを取得するには、Google AI Studio の [API キー](#) ページにアクセスしてください。

API キーは秘密です！他者と共有したり、クライアント側のコードで公開したりしないでください。

この **API キー** はコンポーネントの **GeminiOptions.ApiKey** プロパティに設定する必要があります。

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "YOUR_API_KEY";
```

メッセージ

Gemini モデルにコンテンツを送信し、生成された応答を受信します。モデルは提供された入力に基づいて次のメッセージを生成します。

- **_CreateContent**：指定されたモデルとユーザープロンプトでコンテンツを作成します。
 - **Model**：使用するモデル（例：gemini-2.0-flash）。
 - **Message**：ユーザーメッセージの内容。
 - **MaxOutputTokens**：生成するトークンの最大数。
- **_CreateContentWithSystem**：システム指示を含むコンテンツを作成します。
 - **Model**：使用するモデル。
 - **System**：モデルの動作を設定するシステム指示。
 - **Message**：ユーザーメッセージのコンテンツ。
 - **MaxOutputTokens**：生成するトークンの最大数。
- **_CreateContentStream**：ストリーミング（SSE）を有効にしてコンテンツを作成します。イベントは OnHTTPAPISSE イベントハンドラーを通じて配信されます。

ビジョン

Gemini モデルはテキストプロンプトとともに base64 エンコードされたコンテンツとして渡された画像を理解できます。

- **_CreateVisionContent:** テキストプロンプト付きの画像を送信します。
 - **Model:** 使用するモデル。
 - **Prompt:** 画像に伴うテキストプロンプト。
 - **Base64:** base64 エンコードされた画像データ。
 - **MediaType:** MIME タイプ (image/jpeg、image/png、image/gif、image/webp)。
 - **MaxOutputTokens:** 生成するトークンの最大数。

構造化出力

JSON スキーマを提供して期待されるレスポンス形式を定義することにより、Gemini モデルから構造化された JSON 出力を生成します。

- **_CreateContentJSON:** 構造化された JSON 出力でコンテンツを作成します。
 - **Model:** 使用するモデル。
 - **Message:** ユーザーメッセージのコンテンツ。
 - **Schema:** 期待される出力構造を定義する JSON スキーマ。
 - **MaxOutputTokens:** 生成するトークンの最大数。

モデル

利用可能な Gemini モデルの一覧を取得して詳細を確認します。

- **_GetModels:** 利用可能なすべてのモデルを一覧表示します。
- **_GetModel:** 特定のモデルの詳細を取得します。
 - **ModelId:** 取得するモデルの識別子です。

トークンカウント

モデルに送信する前にメッセージのトークン数を計算します。

- **_CountTokens:** メッセージのトークンをカウントします。
 - **Model:** トークン化に使用するモデル。
 - **Message:** トークンをカウントするテキストコンテンツ。

エンベディング

Gemini モデルを使用してテキストコンテンツのベクトル埋め込みを生成します。

- **_EmbedContent:** テキストのエンベディングを生成します。
 - **Model:** エンベディング生成に使用するモデル。
 - **Text:** エンベディングを生成するテキストコンテンツ。

Gemini | メッセージ

Gemini モデルにプロンプトを送信して生成コンテンツを受け取ります。Gemini API は generateContent エンドポイントを使用してテキスト入力进行处理し、モデルの応答を返します。

簡単な例

Gemini に Hello メッセージを送信します。

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";
ShowMessage(Gemini->_CreateContent("gemini-2.0-flash", "Hello!"));
```

システム命令の使用例

Gemini の動作を制御するシステム指示を含むメッセージを送信します。

```
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?"));
```

ストリーミングの使用例

Server-Sent Events (SSE) を使用して応答をリアルタイムでストリーミングします。OnHTTPAPISSE イベントハンドラを割り当て、ストリーミングイベントを受信してください。

```
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);
}
```

高度な例

コンテンツ生成パラメータを完全に制御するには、型付きのrequest/responseクラスを使用してください。

```
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

Gemini は画像を理解して分析できます。コンテンツパーツ内で base64 エンコードされたデータとして画像を送信できます。

Vision Example

分析のためにbase64エンコードされた画像をGeminiに送信します。

```
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 | モデル

利用可能な Gemini モデルの一覧と情報を取得します。

モデルの一覧

利用可能なすべてのモデルの一覧を表示します。

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";
ShowMessage(Gemini->_GetModels());
```

モデルを取得する

特定のモデルの詳細を取得します。

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";
ShowMessage(Gemini->_GetModel("gemini-2.0-flash"));
```

高度な例

モデルデータの完全な制御には型付きレスポンスクラスを使用してください。

```
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 | 構造化出力

Gemini は、定義したスキーマに準拠した構造化 JSON フォーマットでレスポンスを返すことができます。これにより、スキーマ定義に一致する有効でパース可能な JSON 出力が保証されます。

簡単な例

JSON スキーマ出力でコンテンツを作成するための便利なメソッドを使用します。

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";

String vSchema = "{\"type\":\"object\",\"properties\":{\"name\":{\"type\":\"string\"}, \"age\":{\"type\":\"integer\"}}, \"required\": [\"name\", \"age\"]}";

ShowMessage(Gemini->CreateContentJSON("gemini-2.0-flash",
    "Extract the name and age: John is 30 years old.", vSchema));
```

高度な例

ResponseMimeType と ResponseSchema を使用して構造化出力を設定するには、型付きクラスを使用します。

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";

TsgcGeminiClass_Request_GenerateContent *oRequest = new TsgcGeminiClass_Request_GenerateContent();
try {
    oRequest->Model = "gemini-2.0-flash";
    oRequest->MaxOutputTokens = 4096;
    oRequest->ResponseMimeType = "application/json";
    oRequest->ResponseSchema =
        "{\"type\":\"object\",\"properties\":{\"sentiment\":{\"type\":\"string\"}, \"confidence\":{\"type\":\"number\"}}, \"required\": [\"sentiment\", \"confidence\"]}";

    SetLength(oContents, 1);
    oContents[0] = new TsgcGeminiClass_Request_Content();
    oContents[0]->Role = "user";
    SetLength(oParts, 1);
    oParts[0] = new TsgcGeminiClass_Request_Part();
    oParts[0]->Text = "Analyze the sentiment: I love this product!";
    oContents[0]->Parts = oParts;
    oRequest->Contents = oContents;

    TsgcGeminiClass_Response_GenerateContent *oResponse = Gemini->CreateContent(oRequest);
    try {
        if (Length(oResponse->Candidates) > 0)
            if (Length(oResponse->Candidates[0]->Parts) > 0)
                ShowMessage(oResponse->Candidates[0]->Parts[0]->Text);
    } __finally {
        delete oResponse;
    }
} __finally {
    delete oParts[0];
    delete oContents[0];
    delete oRequest;
}
```

プロパティ

- **ResponseMimeType:** 構造化JSONアウトプットを有効にするには「application/json」に設定します。
- **ResponseSchema:** 期待される出力構造を定義する JSON スキーマ文字列。

Gemini | トークンカウント

API に送信する前にメッセージ内のトークン数をカウントします。コストの見積もりと、メッセージがモデルの制限内に収まることの確認に役立ちます。

簡単な例

メッセージのトークンをカウントします。

```
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?"));
```

高度な例

トークンカウントパラメータを完全に制御するには、型付きのリクエスト/レスポンスクラスを使用してください。

```
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

テキストコンテンツのベクター埋め込みを生成します。埋め込みはテキストの意味的な意味を捉え、類似検索、クラスタリング、その他の NLP タスクに使用できます。

簡単な例

テキストの埋め込みを生成します。

```
TsgcHTTP_API_Gemini *Gemini = new TsgcHTTP_API_Gemini(NULL);
Gemini->GeminiOptions->ApiKey = "API_KEY";
ShowMessage(Gemini->EmbedContent("text-embedding-004", "Hello world"));
```

高度な例

埋め込みデータを完全に制御するために型付き応答クラスを使用します。

```
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 | ツール使用

Gemini は関数呼び出し（ツールの使用）をサポートしており、会話中にモデルが呼び出せる関数を定義できます。Gemini が関数を呼び出すことを決定すると、レスポンスで `functionCall` パートを返します。その後、関数を実行して結果を送り返します。

関数呼び出しフロー

1. 名前、説明、パラメータ（JSON スキーマ）を含む関数宣言を定義します。
2. 関数宣言を含むコンテンツ生成リクエストを送信します。
3. Gemini は、関数名と引数を含む **functionCall** パートで応答します。
4. 提供された引数を使用して関数を実行します。
5. 結果を含む **functionResponse** パートを含む新しいリクエストを送信します。
6. Gemini が最終的な回答で応答します。

使用例

天気関数を定義して、関数呼び出しを処理します。

```
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 =
        "[{\"type\":\"object\", \"properties\": {\"location\": {\"type\":\"string\", \"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;
}
```

プロパティ

- **FunctionDeclarations:** 利用可能な関数を定義する TsgcGeminiClass_Request_FunctionDeclaration の配列。
 - **Name:** 関数名。
 - **Description:** 関数が何をするかの説明。
 - **Parameters:** 関数パラメータを定義する JSON Schema 文字列。
- **ToolChoice:** 関数呼び出しの動作を制御します。モードを設定します（例: 「AUTO」、「ANY」、「NONE」）。

Ollama

Ollama は大規模言語モデルをローカルで実行するためのオープンソースツールです。Llama、Mistral、Gemma、Phi など幅広いモデルをサポートし、クラウド API アクセスを必要とせずにローカル AI 推論を可能にします。sgcWebSocketsライブラリは、Ollama APIと連携するためのDelphiコンポーネント**TsgcHTTP_API_Ollama**を提供します。

Ollama API

Ollama API は、AI搭載アプリケーションのために、ローカルで実行されるモデルへのアクセスを提供します。このAPIは、テキスト生成、ストリーミング、モデル管理、エンベディングをサポートします。モデルはローカルで実行されるため、デフォルトではAPIキーは不要です。ホストとポートは構成可能です。

機能

- メッセージ
 - [Ollama メッセージ使用例](#)
- Models
 - [Ollama Models Examples](#)
- 埋め込み
 - [Ollama 埋め込みの使用例](#)

設定

Ollama はローカルで実行され、デフォルトでは **http://localhost:11434** でリッスンします。**OllamaOptions.Host** プロパティでホストを設定してください。API キーはオプションで、Ollama インスタンスに認証を設定している場合のみ必要です。

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);
Ollama->OllamaOptions->Host = "http://localhost:11434";
```

メッセージ

テキストコンテンツを含む入力メッセージの構造化されたリストを送信すると、モデルは会話の次のメッセージを生成します。

- **_CreateMessage:** 指定したモデルとユーザープロンプトでメッセージを作成します。
 - **Model:** 使用するモデル（例：llama3）。
 - **Message:** ユーザーメッセージの内容。
- **_CreateMessageWithSystem:** システムプロンプト付きのメッセージを作成します。
 - **System:** アシスタントの動作を設定するシステムプロンプト。
- **_CreateMessageStream:** ストリーミング（SSE）を有効にしてメッセージを作成します。イベントは OnHTTPAPISSE イベントハンドラーを通じて配信されます。

モデル

ローカルで利用可能なモデルを管理およびクエリします。

- **_GetModels:** 利用可能なすべてのモデルを一覧表示します。
- **_GetTags:** ローカルで利用可能なすべてのモデルタグを一覧表示します。
- **_ShowModel:** 特定のモデルの詳細情報を取得します。
 - **Model:** クエリするモデルの名前。
- **_PullModel:** Ollama モデルライブラリからモデルをダウンロードします。
 - **Model:** プルするモデルの名前。
- **_DeleteModel:** ローカルで利用可能なモデルを削除します。
 - **Model:** 削除するモデルの名前。

エンベディング

ローカルで実行中のモデルを使用してテスト入力からベクター埋め込みを生成します。

- **_CreateEmbeddings:** 指定されたテキストの埋め込みを生成します。
 - **Model:** 埋め込みに使用するモデル（例: llama3）。
 - **Input:** 埋め込みを生成するテキスト。

TsgcHTTP_API_Ollama

ローカルチャット補完と埋め込み用の Ollama REST API クライアント。

はじめに

ローカルチャット補完と埋め込み用の Ollama REST API クライアント。

はじめに

ローカルチャット補完と埋め込み用の Ollama REST API クライアント。

OllamaOptions プロパティに、APIキーとオプションのHTTP/再試行/ログ設定を構成し、その後、要求オブジェクトとともに `CreateMessage/CreateChatCompletion` などの型付きメソッドを呼び出して、型付きの応答クラスを受け取ります。

プロバイダーの背景については、[Ollama](#) ガイドを参照してください。

リファレンス

- プロパティ — コンポーネントで利用可能な5つのプロパティ。
- メソッド — 2個のパブリックメソッド。
- イベント — 処理できる2つのイベント。

Ollama | メッセージ

テキストコンテンツを含む入力メッセージの構造化されたリストを送信すると、モデルは会話の次のメッセージを生成します。

簡単な例

ローカルの Ollama モデルに Hello メッセージを送信します。

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);
Ollama->OllamaOptions->Host = "http://localhost:11434";
ShowMessage(Ollama->_CreateMessage("llama3", "Hello!"));
```

システムプロンプトの例

モデルの動作を制御するシステムプロンプトを含むメッセージを送信します。

```
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?"));
```

ストリーミングの使用例

Server-Sent Events (SSE) を使用して応答をリアルタイムでストリーミングします。OnHTTPAPISSE イベントハンドラを割り当てて、ストリーミングイベントを受信してください。

```
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 | モデル

ローカルで利用可能なOllamaモデルを管理およびクエリします。

モデルの一覧

利用可能なすべてのモデルの一覧を表示します。

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);
Ollama->OllamaOptions->Host = "http://localhost:11434";
ShowMessage(Ollama->_GetModels());
```

タグの取得

ローカルで利用可能なすべてのモデルタグを一覧表示します。

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);
Ollama->OllamaOptions->Host = "http://localhost:11434";
ShowMessage(Ollama->_GetTags());
```

モデルの表示

特定のモデルに関する詳細情報を取得します。

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);
Ollama->OllamaOptions->Host = "http://localhost:11434";
ShowMessage(Ollama->_ShowModel("llama3"));
```

モデルをプルする

Ollama モデルライブラリからモデルをダウンロードします。

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);
Ollama->OllamaOptions->Host = "http://localhost:11434";
ShowMessage(Ollama->_PullModel("llama3"));
```

Ollama | 埋め込み

ローカルで実行中の Ollama モデルを使用してテキスト入力からベクトル埋め込みを生成します。埋め込みはセマンティック検索、クラスタリング、その他の NLP タスクに使用できます。

エンベディングの作成

指定されたテキスト入力の埋め込みを生成します。

```
TsgcHTTP_API_Ollama *Ollama = new TsgcHTTP_API_Ollama(NULL);  
Ollama->OllamaOptions->Host = "http://localhost:11434";  
ShowMessage(Ollama->_CreateEmbeddings("llama3", "Hello world"));
```

Grok

Grok は xAI が開発した会話型 AI アシスタントで、リアルタイムの知識により役立つ有益な応答を提供するように設計されています。Grok モデルは強力な推論能力と最新情報へのアクセスで知られています。sgcWebSockets ライブラリは Grok API と対話するための Delphi コンポーネント **TsgcHTTP_API_Grok** を提供します。

Grok API

Grok API は AI を活用したアプリケーションを構築するための Grok モデルへのアクセスを提供します。API はテキスト生成、ビジョン (画像理解)、ストリーミング、モデルリステイングをサポートしています。

機能

- メッセージ
 - [Grok メッセージの例](#)
- ビジョン
 - [Grok Visionの使用例](#)
- モデル
 - [Grok モデルの例](#)

設定

Grok API は認証に API キーを使用します。リクエストで使用する API キーを取得するには、[xAI コンソール](#)を訪問してください。

APIキーは秘密です！他者と共有したり、クライアント側のコードで公開したりしないでください。

このAPI Keyは、コンポーネントの**GrokOptions.ApiKey**プロパティで構成する必要があります。

```
TsgcHTTP_API_Grok *Grok = new TsgcHTTP_API_Grok(NULL);
Grok->GrokOptions->ApiKey = "YOUR_API_KEY";
```

メッセージ

テキストコンテンツを含む入力メッセージの構造化されたリストを送信すると、モデルは会話の次のメッセージを生成します。

- **_CreateMessage**：指定されたモデルとユーザープロンプトを使用してメッセージを作成します。
 - **Model**：使用するモデル（例：grok-3）。
 - **Message**：ユーザーメッセージの内容。
- **_CreateMessageWithSystem**: システムプロンプト付きのメッセージを作成します。
 - **System**: アシスタントの動作を設定するシステムプロンプト。
- **_CreateMessageStream**: ストリーミング (SSE) を有効にしてメッセージを作成します。イベントは OnHTTPAPISSE イベントハンドラーを通じて配信されます。

ビジョン

Grok はメッセージ内の Base64 エンコードされたコンテンツとして渡された画像を理解できます。

- **_CreateVisionMessage**: テキストプロンプトと共に画像を送信します。
 - **ImageBase64**: base64エンコードされた画像データ。
 - **MediaType**: MIMEタイプ (image/jpeg、image/png、image/gif、image/webp)。
 - **Prompt**: 画像に添付するテキストプロンプト。

モデル

利用可能な Grok モデルを一覧表示します。

- **_GetModels**: 利用可能なすべてのモデルを一覧表示します。

TsgcHTTP_API_Grok

TsgcHTTP_API_Grok コンポーネント。

はじめに

チャット補完とモデルリストのための xAI Grok REST API クライアント。

はじめに

チャット補完とモデルリストのための xAI Grok REST API クライアント。
APIキーとオプションのHTTP/リトライ/ログ設定を **GrokOptions** プロパティに設定し、リクエストオブジェクトとともに `CreateMessage/CreateChatCompletion` などの型付きメソッドを呼び出して、型付きレスポンスクラスを受け取ります。
プロバイダーの背景については、[Grok](#) ガイドを参照してください。

リファレンス

- プロパティ — コンポーネントで使用可能な 7 個のプロパティ。
- メソッド — 1 個のパブリックメソッド。
- イベント — 処理できるイベントは 0 件です。

Grok | メッセージ

テキストコンテンツを含む入力メッセージの構造化されたリストを送信すると、モデルは会話の次のメッセージを生成します。

簡単な例

Grokにhelloメッセージを送信します。

```
TsgcHTTP_API_Grok *Grok = new TsgcHTTP_API_Grok(NULL);
Grok->GrokOptions->ApiKey = "API_KEY";
ShowMessage(Grok->_CreateMessage("grok-3", "Hello!"));
```

システムプロンプトの例

Grok の動作を制御するシステムプロンプト付きのメッセージを送信します。

```
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?"));
```

ストリーミングの使用例

Server-Sent Events (SSE) を使用して応答をリアルタイムでストリーミングします。OnHTTPAPISSEイベントハンドラを割り当てて、ストリーミングイベントを受信してください。

```
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 | ビジョン

Grokは、メッセージ内のbase64エンコードされたコンテンツとして渡された画像を理解できます。visionエンドポイントを使用して、テキストプロンプトとともに画像を送信し、説明または分析を受け取ります。

Vision Example

分析のために Grok に base64 エンコードされた画像を送信します。

```
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 | モデル

利用可能な Grok モデルとその基本情報を一覧表示します。

モデル一覧の使用例

利用可能なすべての Grok モデルの一覧を取得します。

```
TsgcHTTP_API_Grok *Grok = new TsgcHTTP_API_Grok(NULL);  
Grok->GrokOptions->ApiKey = "API_KEY";  
ShowMessage(Grok->_GetModels);
```

Mistral AI

Mistral AI は効率的で強力な言語モデルを開発するフランスの人工知能企業です。そのモデルは推論、コーディング、多言語タスクにわたる強力なパフォーマンスで知られており、競争力のある効率性を維持しています。sgcWebSocketsライブラリは、Mistral AI APIと対話するためのDelphiコンポーネントTsgcHTTP_API_Mistralを提供します。

Mistral API

Mistral APIは、AIを活用したアプリケーションを構築するためのMistralモデルへのアクセスを提供します。APIは、テキスト生成、ビジョン（画像理解）、ストリーミング、埋め込み、JSONモード、モデルリストをサポートしています。

機能

- メッセージ
 - [Mistral メッセージの使用例](#)
- ビジョン
 - [Mistral ビジョンの使用例](#)
- モデル
 - [Mistral モデルの使用例](#)
- エンベディング
 - [Mistral エンベディングの使用例](#)

設定

Mistral API は認証に API キーを使用します。Mistral Console の [API キー](#) ページにアクセスして、リクエストで使用する API キーを取得します。APIキーは秘密です！他者と共有したり、クライアント側のコードで公開したりしないでください。このAPI Keyは、コンポーネントのMistralOptions.ApiKeyプロパティで構成する必要があります。

```
TsgcHTTP_API_Mistral *Mistral = new TsgcHTTP_API_Mistral(NULL);  
Mistral->MistralOptions->ApiKey = "YOUR_API_KEY";
```

メッセージ

テキストコンテンツを含む入力メッセージの構造化されたリストを送信すると、モデルは会話の次のメッセージを生成します。

- **_CreateMessage:** 指定したモデルとユーザープロンプトでメッセージを作成します。
 - **Model:** 使用するモデル（例：mistral-large-latest）。
 - **Message:** ユーザーメッセージの内容。
- **_CreateMessageWithSystem:** システムプロンプト付きのメッセージを作成します。
 - **System:** アシスタントの動作を設定するシステムプロンプト。
- **_CreateMessageStream:** ストリーミング（SSE）を有効にしてメッセージを作成します。イベントは OnHTTPAPISSE イベントハンドラーを通じて配信されます。
- **_CreateMessageJSON:** JSONモードを有効にしてメッセージを作成します。モデルは有効なJSON応答を返します。
 - **Model:** 使用するモデルです。
 - **Message:** ユーザーメッセージの内容です。

ビジョン

Mistral はメッセージ内のbase64エンコードコンテンツとして渡された画像を理解できます。

- **_CreateVisionMessage:** テキストプロンプトと共に画像を送信します。
 - **ImageBase64:** base64エンコードされた画像データ。
 - **MediaType:** MIMEタイプ（image/jpeg、image/png、image/gif、image/webp）。

- **Prompt:** 画像に添付するテキストプロンプト。

モデル

利用可能な Mistral モデルを一覧表示します。

- **_GetModels:** 利用可能なすべてのモデルを一覧表示します。

エンベディング

セマンティック検索、クラスタリング、その他の機械学習タスクに使用できる、指定された入力のベクター表現を取得します。

- **_CreateEmbeddings:** 入力テキストを表すエンベディングベクターを作成します。
 - **Model:** 使用するモデル（例：mistral-embed）。
 - **Input:** エンベディングを取得する入力テキスト。

TsgcHTTP_API_Mistral

チャット補完と埋め込み用の Mistral REST API クライアント。

はじめに

チャット補完と埋め込み用の Mistral REST API クライアント。

はじめに

チャット補完と埋め込み用の Mistral REST API クライアント。

MistralOptions プロパティにAPIキーおよびオプションのHTTP/リトライ/ログ設定を構成し、リクエストオブジェクトを指定して `CreateMessage/CreateChatCompletion` などの型付きメソッドを呼び出し、型付きレスポンスクラスを受け取ります。

プロバイダーの背景については、[Mistral](#) ガイドを参照してください。

リファレンス

- プロパティ — コンポーネントで利用可能なプロパティ 5 件。
- メソッド — 2 つのパブリックメソッド。
- イベント — 処理可能な 2 つのイベント。

Mistral | メッセージ

テキストコンテンツを含む入力メッセージの構造化されたリストを送信すると、モデルは会話の次のメッセージを生成します。

簡単な例

Mistral にハローメッセージを送信します。

```
TsgcHTTP_API_Mistral *Mistral = new TsgcHTTP_API_Mistral(NULL);
Mistral->MistralOptions->ApiKey = "API_KEY";
ShowMessage(Mistral->_CreateMessage("mistral-large-latest", "Hello!"));
```

システムプロンプトの例

Mistral の動作を制御するシステムプロンプトを含むメッセージを送信します。

```
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?"));
```

ストリーミングの使用例

Server-Sent Events (SSE) を使用して応答をリアルタイムでストリーミングします。OnHTTPAPISSE イベントハンドラを割り当てて、ストリーミングイベントを受信してください。

```
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

Mistral に JSON 形式のレスポンスをリクエストします。

```
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 | ビジョン

Mistral はメッセージ内のbase64エンコードコンテンツとして渡された画像を理解できます。ビジョンエンドポイントを使用して、テキストプロンプトと共に画像を送信し、説明や分析を受信します。

Vision Example

Mistral に base64 エンコードされた画像を送信して分析します。

```
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 | モデル

利用可能なMistralモデルとその基本情報をリストします。

モデル一覧の使用例

利用可能なすべての Mistral モデルのリストを取得します。

```
TsgcHTTP_API_Mistral *Mistral = new TsgcHTTP_API_Mistral(NULL);
Mistral->MistralOptions->ApiKey = "API_KEY";
ShowMessage(Mistral->_GetModels);
```

Mistral | 埋め込み

セマンティック検索、クラスタリング、その他の機械学習タスクに使用できる、指定された入力のベクター表現を取得します。

埋め込みの使用例

テキスト入力のembeddingベクトルを作成します。

```
TsgcHTTP_API_Mistral *Mistral = new TsgcHTTP_API_Mistral(NULL);
Mistral->MistralOptions->ApiKey = "API_KEY";
ShowMessage(Mistral->_CreateEmbeddings("mistral-embed", "Hello world"));
```

IoT

IoT（モノのインターネット）とは、パーソナルコンピューターやモバイルデバイスなどの従来のコンピューティングプラットフォームを超え、従来「スマートではない」またはインターネットに接続されていない物理的なデバイスや日常的なオブジェクトまで、インターネット接続を拡張するコンセプトを指します。電子機器、インターネット接続、その他のハードウェア（センサーなど）が組み込まれたこれらのデバイスは、インターネットを介して互いに通信・連携し、遠隔から監視・制御することができます。

sgcWebSockets パッケージは以下の IoT クライアントを実装しています：

1. Amazon AWS IoT: AWS IoT は、センサー、アクチュエーター、組み込みマイクロコントローラー、スマート家電などのインターネット接続デバイスと AWS クラウド間の安全な双方向通信を提供します。これにより、複数のデバイスからテレメトリデータを収集し、データを保存して分析できます。また、ユーザーがスマートフォンやタブレットからこれらのデバイスを制御できるアプリケーションを作成することもできます。

2. Azure IoT Hub: IoT Hubは、クラウドでホストされる マネージドサービスで、IoTアプリケーションとそれが管理するデバイス間の 双方向通信のための中央メッセージハブとして機能します。Azure IoT Hubを使用して、数百万のIoTデバイスと クラウドホストのソリューションバックエンド間で信頼性の高いセキュアな通信を備えた IoT ソリューションを構築できます。

IoT Amazon MQTT クライアント

AWS IoT とは何ですか？

AWS IoT は、センサー、アクチュエーター、組み込みマイクロコントローラー、スマートアプライアンスなどのインターネット接続デバイスと AWS クラウドとの間に安全な双方向通信を提供します。これにより、複数のデバイスからテレメトリデータを収集してデータを保存および分析することができます。また、ユーザーが電話やタブレットからこれらのデバイスを制御できるアプリケーションを作成することもできます。

Message broker

デバイスと AWS IoT アプリケーションが互いにメッセージを送受信するための安全なメカニズムを提供します。MQTT プロトコルを直接使用することも、WebSocket 経由の MQTT を使用してパブリッシュとサブスクライブを行うこともできます。

AWS IoT メッセージブローカーは、AWS IoT との間でメッセージを送受信できるパブリッシュ/サブスクライブブローカーサービスです。AWS IoT と通信する場合、クライアントは `Sensor/temp/room1` のようなトピックに宛てたメッセージを送信します。

メッセージブローカーは次に、そのトピックのメッセージを受信するために登録したすべてのクライアントにメッセージを送信します。メッセージを送信する行為をパブリッシングと呼びます。トピックフィルターのメッセージを受信するために登録する行為をサブスクライビングと呼びます。

トピック名前空間は AWS アカウントとリージョンのペアごとに分離されています。たとえば、AWS アカウントの `Sensor/temp/room1` トピックは別の AWS アカウントの `Sensor/temp/room1` トピックとは独立しています。リージョンも同様です。同じ AWS アカウントで `us-east-1` の `Sensor/temp/room1` トピックは `us-east-2` の同じトピックとは独立しています。AWS IoT は AWS アカウントやリージョンをまたいだメッセージの送受信をサポートしていません。

メッセージブローカーは、すべてのクライアントセッションと各セッションのサブスクリプションのリストを維持します。トピックにメッセージが発行されると、ブローカーはそのトピックにマッピングされるサブスクリプションを持つセッションを確認します。次にブローカーは、現在接続中のクライアントを持つすべてのセッションに発行メッセージを転送します。

MQTT クライアント

`TsgcIoTAmazon_MQTT_Client` は AWS IoT への接続に使用するコンポーネントです。1つのクライアントは1つのデバイスにのみ接続できます。クライアントはプレーン MQTT プロトコルを使用して接続し、X.509 クライアント証明書を使用して認証します。

AWS IoT に接続するには、クライアントに以下のプロパティが必要です：

Amazon.ClientId: クライアントの識別子。オプションです。

Amazon.Endpoint: MQTT クライアントが接続するサーバー名。

Amazon.Port: デフォルトではポート 8883 を使用します。ポートが 443 の場合、接続に ALPN を自動的に使用します(カスタム Indy バージョンが必要)。

AWS IoT Core は MQTT および WebSocket Secure (WSS) プロトコルを使用してメッセージをパブリッシュおよびサブスクライブするデバイスとクライアントをサポートしています。次の表は AWS IoT デバイスエンドポイントがサポートするプロトコルと使用する認証メソッドおよびポートを示しています。

プロトコル	認証	Port	ALPN プロトコル名
MQTT over WebSocket	署名バージョン4	443	

MQTT over WebSocket	カスタム認証	443	
MQTT	X.509クライアント証明書	443	x-amzn-mqtt-ca
MQTT	X.509クライアント証明書	8883	
MQTT	カスタム認証	443	mqtt

証明書による認証

Amazon AWS コンソールで証明書を作成し、証明書が保存されているパスを設定する必要があります。

IOHandler として **OpenSSL** を使用する場合、以下のパスに証明書を設定する必要があります

Certificate.Enabled: 証明書を使用する場合は True に設定します。

Certificate.CertFile: X.509クライアント証明書へのパス。

Certificate.KeyFile: X.509クライアントキーファイルへのパスです。

SChannel を IOHandler として使用する場合、まず PEM 証明書とキーを PFX 証明書に変換します。これには OpenSSL バイナリが必要です。

```
openssl pkcs12 -inkey 884ccf73ff-private.pem.key -in 884ccf73ff-certificate.pem.crt -export -out 884ccf73ff-cert.pfx
```

次に以下のパスを設定してください（証明書にはすでにキーファイルが含まれているため、キーファイルを設定する必要はありません）。

Certificate.Enabled: 証明書を使用する場合は True に設定します。

Certificate.CertFile: PFX証明書へのパス

SignatureV4 Authentication

Amazon AWS コンソールでユーザーを作成し、WebSocket リクエストの署名に使用するアクセスキーとシークレットキーを保存する必要があります。

SignatureV4.Enabled: この認証タイプを使用する場合は True に設定します。

SignatureV4.Region: デバイスが配置されているリージョン（例：us-east-1）。

SignatureV4.AccessKey: Amazon コンソールで作成されたか、一時的な認証情報として取得されたアクセスキー。

SignatureV4.SecretKey: Amazon コンソールで作成またはテンポラリー認証情報として取得したシークレットキー。

SignatureV4.SessionToken: （条件付き）一時セキュリティ認証情報を使用している場合は、ここにセキュリティトークンを設定します。

OpenSSL_Options: OpenSSL ライブラリの設定。

APIVersion: 使用する OpenSSL API を定義できます。

oslAPI_1_0: API 1.0 OpenSSL を使用します。Indy で最後にサポートされたバージョンです。

oslAPI_1_1: API 1.1のOpenSSLを使用します。カスタムIndyライブラリが必要で、OpenSSL 1.1.1ライブラリ（TLS 1.3サポート付き）の使用が可能になります。

oslAPI_3_0: API 3.0 OpenSSLを使用します。当社のカスタムIndyライブラリが必要であり、OpenSSL 3.0.0ライブラリの使用を可能にします(TLS 1.3サポート付き)。

LibPath: opensslライブラリの場所を設定できます

oslpNone: これはデフォルトです。opensslライブラリは、バイナリと同じフォルダー、または既知のパスにある必要があります。

oslpDefaultFolder: すべてのIDEパーソナリティに対してライブラリが配置されるべきopensslパスを自動的に設定します。

oslpCustomFolder: このオプションが選択されている場合、プロパティ `LibPathCustom` にフルパスを定義します。

LibPathCustom: `LibPath = oslpCustomFolder` の場合、`openssl` ライブラリが格納されているフォルダーの完全パスをここで定義します。

UnixSymLinks: Unix システムでシンボリックリンクの読み込みを有効または無効にします（デフォルトでは有効です。OSX64 は除きます）。

oslsSymLinksDefault: デフォルトでは有効です。ただし OSX64 では無効です（MacOS Monterey 以降、バージョンなしでライブラリを読み込もうとすると失敗するため）。

oslsSymLinksLoadFirst: `SymLinks` を読み込み、バージョンライブラリを読み込む前に行います。

oslsSymLinksLoad: バージョンライブラリの読み込みを試みた後にシンボリックリンクを読み込みます。

oslsSymLinksDontLoad: `SymLinks` を読み込みません。

*SignatureV4 には Indy 10.5.7 以降が必要です

カスタム認証

カスタム認証を使用すると、オーソライザーリソースを使用してクライアントの認証と認可方法を定義できます。デバイスは、リクエストのヘッダーフィールドまたはクエリパラメーター（WebSocket 上の MQTT プロトコルの場合）、あるいは MQTT CONNECT メッセージのユーザー名とパスワードフィールド（MQTT および WebSocket 上の MQTT プロトコルの場合）で認証情報を渡します。

CustomAuthentication.Enabled: この種類の認証を使用する場合は `True` に設定します。

CustomAuthentication.Parameters: サーバーに渡されるクエリ パラメータをここで設定します（デフォルトは `/mqtt`）

CustomAuthentication.Headers: カスタムヘッダーフィールドをここに設定できます。

CustomAuthentication.WebSockets: `true` に設定すると、接続は WebSocket プロトコルで動作します。それ以外の場合はプレーン TCP で動作します。

MQTTAuthentication.Enabled: MQTT 接続でユーザー名/パスワードを渡す必要がある場合は、このプロパティを有効にします。

MQTTAuthentication.Username: mqtt 接続のユーザー名です

MQTTAuthentication.Password: MQTT 接続のシークレット。

クライアントはオプションで `ClientId` を送信してクライアント接続を識別できます。他のクライアントはこのクライアントが接続、サブスクライブ、切断するたびに通知を受け取るためにサブスクライブできます。

認証

ポート 8883 を使用して TCP をトランスポートとして接続できない場合（デフォルト）、Amazon は「AWS IoT Core ポリシー」を使用してクライアントとサブスクリプションに認証を提供または拒否します。おそらくクライアント ID を認可する必要があります。

Amazon AWS コンソールにログインし、IoT Core に移動して「Secure/Policies」メニューにアクセスし、IoT Thing にアタッチされているポリシーを選択して、最後に接続がどのように設定されているか確認してください。

例：

```
{
  "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-*"
  ]
}
```

この設定は、ID が `sdk-java`、`basicPubSub`、`sdk-nodejs-*` のクライアントのみが接続できることを意味します。必要に応じて変更して再試行してください。

それでも動作しない場合は、ログを有効にして CloudWatch で接続できない理由を確認します。

その他のプロパティ

MQTTHeartBeat: 有効にすると、X秒ごとにpingを送信してMQTT接続を維持しようとします。

Interval: 各 ping 間の秒数。

MQTTAuthentication: 有効にすると MQTT 接続にユーザー名とパスワードを含めます。

UserName: ユーザーの名前

パスワード: シークレット文字列

WatchDog: 有効にすると、予期しない切断が検出されたときに、自動的にサーバーへの再接続を試みます。

Interval: 再接続試行前の秒数。

Attempts: 再接続試行の最大回数。ゼロは無制限を意味します。

LogFile: 有効にすると、ソケットメッセージをログファイルに保存します（デバッグに便利）。複数のスレッドからアクセスされる場合、ログファイルへのアクセスはスレッドセーフではありません。

Enabled: 有効にすると、ソケットでメッセージが受信・送信されるたびにファイルに保存されます。

FileName: ファイル名への完全パス。

Implementation

Amazon の MQTT 実装は MQTT バージョン 3.1.1 をベースにしていますが、仕様から以下の点で異なります:

- AWS IoT では、Quality of Service (QoS) 0 のトピックにサブスクライブするということは、メッセージがゼロ回以上配信されることを意味します。メッセージが複数回配信される場合があります。複数回配信されるメッセージは異なるパケット ID で送信される場合があります。これらの場合、DUP フラグは設定されません。
- AWS IoT は QoS 2 でのパブリッシュおよびサブスクライブをサポートしていません。AWS IoT メッセージブローカーは QoS 2 がリクエストされた場合に PUBACK や SUBACK を送信しません。
- 接続リクエストに応答する際、メッセージブローカーはCONNACKメッセージを送信します。このメッセージには、接続が以前のセッションを再開しているかどうかを示すフラグが含まれています。同じクライアントIDを持つ2つのMQTTクライアントが同時に接続する場合、このフラグの値が正しくない可能性があります。
- クライアントがトピックをサブスクライブする際、ブローカーが SUBACK を送信した時点から新しい一致するメッセージの受信が開始されるまでの間に遅延が発生する場合があります。
- MQTT 仕様では、パブリッシャーがブローカーにトピックに送信された最後のメッセージを保持し、将来のすべてのトピック購読者に送信するよう要求する規定があります。AWS IoT は保持メッセージをサポートしていません。メッセージを保持するリクエストが行われると、接続が切断されます。
- メッセージブローカーはクライアントIDを使用して各クライアントを識別します。クライアントIDはMQTTペイロードの一部としてクライアントからメッセージブローカーに渡されます。同じクライアントIDを持つ2つのクライアントをメッセージブローカーに同時に接続することはできません。別のクライアントが使用しているクライアントIDを使用してクライアントがメッセージブローカーに接続すると、両方のクライアントにCONNACKメッセージが送信され、現在接続されているクライアントは切断されます。
- まれに、メッセージブローカーが異なるパケット ID で同じ論理 PUBLISH メッセージを再送信することがあります。
- メッセージブローカーはメッセージとACKが受信される順序を保証しません。

AWS IoT への接続

まず、AWS コンソールにサインインし、新しいデバイスを登録してデバイス用の X.509 証明書を作成します。完了したら、新しい TsgcloTAmazon_MQTT_Client を作成して AWS IoT サーバーに接続できます。例:


```
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");
}
```

トピック

メッセージブローカーはトピックを使用してパブリッシングクライアントからサブスクライビングクライアントにメッセージをルーティングします。スラッシュ (/) はトピック階層を区切るために使用されます。次の表は購読時のトピックフィルターで利用できるワイルドカードの一覧です。# は購読するトピックの最後の文字である必要があります。現在のツリーとすべてのサブツリーに一致するワイルドカードとして機能します。

たとえば、Sensor/# へのサブスクリプションは Sensor/、Sensor/temp、Sensor/temp/room1 に発行されたメッセージを受信しますが、Sensor に発行されたメッセージは受信しません。

+ トピック階層内の 1 つのアイテムに正確にマッチします。例えば、Sensor+/room1 へのサブスクリプションは Sensor/temp/room1、Sensor/moisture/room1 などに発行されたメッセージを受信します。

```
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);
}
```

Reserved Topics

以下のメソッドは予約済みトピックへのサブスクライブ/公開に使用されます。

Subscribe_ClientConnected(const aClientId: String): 指定したクライアント ID を持つ MQTT クライアントが AWS IoT に接続すると、AWS IoT はこのトピックにパブリッシュします

Subscribe_ClientDisconnected(const aClientId: String): 指定したクライアント ID を持つ MQTT クライアントが AWS IoT から切断すると、AWS IoT がこのトピックに発行します

Subscribe_ClientSubscribed(const aClientId: String): 指定したクライアント ID を持つ MQTT クライアントが MQTT トピックにサブスクライブすると、AWS IoT がこのトピックに発行します

Subscribe_ClientUnSubscribed(const aClientId: String): 指定されたクライアント ID を持つ MQTT クライアントが MQTT トピックからサブスクライブ解除したときに、AWS IoT がこのトピックにパブリッシュします。

Publish_Rule(const aRuleName, aText: String): デバイスまたはアプリケーションがこのトピックに公開することで、ルールを直接トリガーします。

Publish_DeleteShadow(const aThingName, aText: String): デバイスまたはアプリケーションがシャドウを削除するためにこのトピックに公開します。

Subscribe_DeleteShadow(const aThingName: String): デバイスまたはアプリケーションがシャドウを削除するためにこのトピックにサブスクライブします

Subscribe_ShadowDeleted(const aThingName: String): シャドウが削除されたときに Device Shadow サービスがこのトピックにメッセージを送信します

Subscribe_ShadowRejected(const aThingName: String): シャドウの削除リクエストが拒否された場合、Device Shadow サービスはこのトピックにメッセージを送信します。

Publish_ShadowGet(const aThingName, aText: String): アプリケーションまたはシャドウを取得するためにこのトピックに空のメッセージをパブリッシュします

Subscribe_ShadowGet(const aThingName: String): アプリケーションまたはシングルがシャドウを取得するためにこのトピックにサブスクライブします。

Subscribe_ShadowGetAccepted(const aThingName: String): Device Shadowサービスは、shadowのリクエストが正常に行われたときにこのトピックにメッセージを送信します

Subscribe_ShadowGetRejected(const aThingName: String): Device Shadow サービスがシャドウへのリクエストを拒否したときにこのトピックにメッセージを送信します

Publish_ShadowUpdate(const aThingName, aText: String): thingまたはアプリケーションがこのトピックにパブリッシュしてシャドウを更新します

Subscribe_ShadowUpdateAccepted(const aThingName: String) : Device Shadow サービスは、シャドウへの更新が正常に行われたときにこのトピックにメッセージを送信します

Subscribe_ShadowUpdateRejected(const aThingName: String): Device Shadowサービスは、shadowへの更新が拒否されたときにこのトピックにメッセージを送信します

Subscribe_ShadowUpdateDelta(const aThingName: String) : デバイスシャドウサービスは、シャドウの reported セクションと desired セクションの間に差分が検出されたとき、このトピックにメッセージを送信します

Subscribe_ShadowUpdateDocuments(const aThingName: String): シャドウへの更新が正常に実行されるたびに、AWS IoT はこのトピックに状態ドキュメントをパブリッシュします

永続セッション

永続的なセッションは MQTT メッセージブローカーへの継続的な接続を表します。クライアントが永続的なセッションを使用して AWS IoT メッセージブローカーに接続すると、メッセージブローカーは接続中にクライアントが行うすべてのサブスクリプションを保存します。クライアントが切断すると、メッセージブローカーは未確認の QoS 1 メッセージと、クライアントがサブスクライブしているトピックに発行された新しい QoS 1 メッセージを保存します。クライアントが永続的なセッションに再接続すると、すべてのサブスクリプションが復元され、保存されたすべてのメッセージが最大秒速10メッセージの速度でクライアントに送信されます。

OnMQTTBeforeConnect イベントで **cleanSession** パラメータを **False** に設定することで、MQTT 永続セッションを作成します。クライアントのセッションが存在しない場合は、新しい永続セッションが作成されます。セッションが既に存在する場合は再開されます。

デバイスは **OnMQTTConnect** イベントの **Session** 属性を確認して、永続セッションが存在するかどうかを判断する必要があります。**Session** が **True** の場合、永続セッションが存在し、保存されたメッセージがクライアントに配信されます。**Session** が **False** の場合、永続セッションは存在せず、クライアントはトピックフィルターに再サブスクライブする必要があります。

永続セッションのデフォルトの有効期限は1時間です。有効期限はメッセージブローカーがクライアントの切断 (MQTT 切断またはタイムアウト) を検出した時点から始まります。永続セッションの有効期限は標準の制限引き上げプロセスを通じて延長できます。クライアントが有効期限内にセッションを再開しなかった場合、セッションは終了し、関連する保存済みメッセージは破棄されます。有効期限は概算であり、セッションは設定した期間より最大30分長く (ただし短くはならず) 永続化される場合があります。

一時的な認証情報

AWS IoT Coreはアイデンティティプールを通じて取得した一時的な認証情報で動作できます。アイデンティティには2種類あります。

- **UnAuthenticated:** IAMでポリシータイプを設定するだけで済みます
- **認証済み:** IAMおよびAWS IoT Coreポリシーでポリシータイプを設定する必要があります

未認証

未認証の認証情報を使用している場合は、IAM メニューで自動的に作成された未認証ロールにポリシーをアタッチするだけです。次に、Cognito サービスが返した Access キー、Secret Key、Token を設定してクライアントを設定します。

未認証の認証情報を取得するための .NET コードを以下に示します

```
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;
```

認証済み

認証済み資格情報。IAM メニューで自動作成される認証済みロールにポリシーをアタッチし、AWS IoT Core ポリシーのユーザーポリシーをアタッチする必要があります。

IoT Core ポリシーメニューで新しいポリシーを作成し、新しいユーザーが認証するたびにこのポリシーをそのユーザーにアタッチします。

AWS の以下のコマンドを使用して、ポリシーをアタッチしたり Lambda 関数を作成したりできます。

```
aws iot attach-policy --policy-name PolicyName --target us-east-1:XXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXXXXXX
```

デバイスプロビジョニング

Fleet Provisioning サービスは以下の MQTT API 操作をサポートしています。

- **CreateCertificateFromCsr:** 証明書署名リクエスト（CSR）から証明書を作成します。AWS IoT は Amazon ルート認証局（CA）によって署名されたクライアント証明書を提供します。新しい証明書は PENDING_ACTIVATION ステータスになります。この証明書でデバイスをプロビジョニングするために RegisterThing を呼び出すと、証明書のステータスはテンプレートに従って ACTIVE または INACTIVE になります。
- **CreateKeysAndCertificate:** 新しいキーと証明書を作成します。AWS IoT は Amazon ルート認証局（CA）によって署名されたクライアント証明書を提供します。新しい証明書のステータスは PENDING_ACTIVATION です。RegisterThing を呼び出してこの証明書でモノをプロビジョニングすると、テンプレートの説明に従って証明書のステータスが ACTIVE または INACTIVE に変わります。
- **RegisterThing:** 事前定義されたテンプレートを使用してデバイスをプロビジョニングします。

CreateCertificateFromCsr

CertificateSigningRequest をパラメータとして渡して CreateCertificateFromCsr メソッドを使用し、証明書を作成します。このリクエストへのレスポンスを受信するには、まず以下のメソッドにサブスクライブします: SubscribeCreateCertificateFromCsrResponse と SubscribeCreateCertificateFromCsrError。

CreateKeysAndCertificate

CreateKeysAndCertificate メソッドを使用して新しい証明書とキーを作成します。このリクエストへのレスポンスを受信するには、最初に以下のメソッドにサブスクライブします: SubscribeCreateKeysAndCertificateResponse および SubscribeCreateKeysAndCertificateError。

RegisterThing

RegisterThingメソッドを使用して、Template NameとJSON形式のPayloadをパラメータとして渡し、新しいthingを登録します。このリクエストへの応答を受信するには、まず次のメソッドSubscribeRegisterThingResponseとSubscribeRegisterThingErrorをサブスクライブします。

IoT Azure MQTT クライアント

Azure IoT Hub とは？

IoT Hub は、IoT アプリケーションとその管理下にあるデバイス間の双方向通信のための中央メッセージハブとして機能する、クラウドでホストされるマネージドサービスです。Azure IoT Hub を使用すると、数百万台の IoT デバイスとクラウドでホストされるソリューションバックエンド間の信頼性が高くセキュアな通信で IoT ソリューションを構築できます。事実上あらゆるデバイスを IoT Hub に接続できます。

IoT Hub はデバイスからクラウドへ、クラウドからデバイスへの双方向通信をサポートしています。IoT Hub は、デバイスからクラウドへのテレメトリ、デバイスからのファイルアップロード、クラウドからデバイスを制御するためのリクエスト/レスポンスメソッドなど、複数のメッセージングパターンをサポートしています。IoT Hub のモニタリングは、デバイスの作成、デバイスの障害、デバイスの接続などのイベントを追跡することで、ソリューションの健全性を維持するのに役立ちます。

IoT Hub の機能は、製造で使用する産業機器の管理、医療での貴重な資産の追跡、オフィスビルの使用状況の監視など、スケーラブルで完全な機能を持つ IoT ソリューションを構築するのに役立ちます。

Message broker

IoT Hub はデバイスがデータを送信するためのセキュアな通信チャンネルを提供します。IoT Hub とデバイス SDK はデバイス接続のために以下のプロトコルをサポートしています。

- **MQTT**
- **MQTT over WebSockets**

複数の認証タイプがさまざまなデバイス機能をサポートします：

- **SAS** トークンベースの認証により、IoT ソリューションをすぐに開始できます。
- 安全で標準に基づいた認証のための、個別の **X.509 証明書** 認証。

MQTT クライアント

TsgcIoTAzure_MQTT_Client は Azure IoT に接続するために使用するコンポーネントです。1 つのクライアントは 1 つのデバイスにのみ接続できます。クライアントはプレーン MQTT プロトコルを使用して接続し、SAS / X.509 クライアント証明書を使用して認証します。

Azure IoT Hub に接続するには、クライアントに次のプロパティが必要です：

Azure.IoTHub: MQTT クライアントが接続するサーバー名です。
Azure.Deviceld: Azure IoT Hub 内のデバイス名。

Azure は複数の認証タイプに対応しており、デフォルトでは SAS トークンを使用します。

SAS 認証

SAS.Enabled: 認証が SAS を使用する場合は有効にします。
SAS.SecretKey: Azure IoT アカウントからの SAS トークン。
SAS.KeyName: 共有アクセスキー名。
SAS.Expiry: SAS トークンが期限切れになるまでの分数を設定します。デフォルト値は 1440（24 時間）です。

接続文字列を持っている場合は、**ReadConnectionString** メソッドを使用して接続文字列の値を自動的に読み取ることができます。例：

```
ReadConnectionString('HostName=yourhub.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=Yj7RRPnkSDTv+UCFLgwIP/FrbDymZv4qVAIoTLHUFR8=');
```

X509 証明書

OpenSSLをIOHandlerとして使用する

Certificate.Enabled : 認証が証明書を使用する場合に有効にします。

Certificate.CertFile: X.509クライアント証明書へのパス。

Certificate.KeyFile: X.509クライアントキーファイルへのパスです。

Certificate.Password: 証明書にパスワードがある場合は、ここに設定します。

Version: TLSバージョン。デフォルトではTLS 1.0を使用します

IOHandler として **SChannel** を使用する

Certificate.Enabled : 認証が証明書を使用する場合に有効にします。

Certificate.CertFile: PFX 証明書へのパス（まず証明書を PFX に変換する必要があります）。[詳細を読む](#)。

Certificate.Password: 証明書にパスワードがある場合は、ここに設定します。

Version: TLSバージョン。デフォルトではTLS 1.0を使用します

その他のプロパティ:

MQTTHeartBeat: 有効にすると、X秒ごとにpingを送信してMQTT接続を維持しようとします。

Interval: 各 ping 間の秒数。

WatchDog: 有効にすると、予期しない切断が検出されたときに、自動的にサーバーへの再接続を試みます。

Interval: 再接続試行前の秒数。

Attempts: 再接続試行の最大回数。ゼロは無制限を意味します。

LogFile: 有効にすると、ソケットメッセージをログファイルに保存します（デバッグに便利）。複数のスレッドからアクセスされる場合、ログファイルへのアクセスはスレッドセーフではありません。

Enabled: 有効にすると、ソケットでメッセージが受信・送信されるたびにファイルに保存されます。

FileName: ファイル名への完全パス。

Azure の MQTT 実装は MQTT バージョン 3.1.1 に基づいていますが、以下の点で仕様から逸脱しています。

- IoT Hub は QoS 2 メッセージをサポートしません。デバイス アプリが QoS 2 でメッセージをパブリッシュすると、IoT Hub はネットワーク接続を閉じます。
- IoT Hub はRetainメッセージを永続化しません。デバイスがRETAINフラグを1に設定してメッセージを送信すると、IoT Hub はx-opt-retainアプリケーションプロパティをメッセージに追加します。この場合、IoT Hub はRetainメッセージを永続化する代わりに、バックエンドアプリに渡します。
- IoT Hub はデバイスごとに1つのアクティブな MQTT 接続のみをサポートします。同じデバイス ID に対して新しい MQTT 接続が行われると、IoT Hub は既存の接続を切断します。

Azure IoT Hub への接続

まず、Azure アカウントにサインインし、新しいデバイスを登録して、このデバイスの認証方法を作成する必要があります。それが完了したら、新しい TsgcIoTAzure_MQTT_Client を作成して Azure IoT Hub に接続できます。

例:

```
TsgcIoTAzure_MQTT_Client *oClient = new TsgcIoTAzure_MQTT_Client();
oClient->Azure->IoTHub = "youriothub.azure-devices.net";
oClient->Azure->DeviceId = "YourDeviceId";
oClient->SAS->Enabled = true;
oClient->SAS->SecretKey = "YourSecretKey";
```

```
oClient->OnMQTTConnect = OnMQTTConnectEvent;
oClient->Active = true;

private void OnMQTTConnect(TsgcWSConnection *Connection, const bool Session, const TmqttConnReturnCode *ReturnCode)
{
    ShowMessage("Connected to Azure IoT Hub");
}
```

デバイスからクラウドへ

デバイスアプリからソリューションバックエンドに情報を送信する場合、IoT Hub は次のオプションを提供します。

1. 時系列のテレメトリとアラートのためのデバイスからクラウドへのメッセージ。

```
oClient->Send_DeviceToCloud("{\"temp\": 10}", azuIoTQoS1);
```

TStringList を使用してキーと値のプロパティを送信できます。TStringList に必要なメッセージプロパティを入力して引数として渡します。

```
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();
}
```

メッセージの **ContentType** と **ContentEncoding** を設定する必要がある場合は、プロパティリストにこれらの値を追加する必要があります。これらのプロパティの名前は Azure によって定義されています。

名前	値
\$.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. デバイスツインのreportedプロパティは、利用可能な機能、条件、または長時間実行ワークフローの状態などのデバイス状態情報を報告するためのものです。例えば、設定やソフトウェアのアップデートなどです。

```
oClient->Set_DeviceTwinsProperties("1", "{\"sgc\":1}");
```

クラウドからデバイスへ

IoT Hub はデバイスアプリがバックエンドアプリに機能を公開するための3つのオプションを提供しています：

1. **ダイレクトメソッド**は結果の即時確認が必要な通信に使用します。ダイレクトメソッドは、ファンのオン/オフなど、デバイスのインタラクティブな制御によく使用されます。

```
oClient->Subscribe_DirectMethod();
```

次のメソッドを使用してパブリックメソッドに返答できます。

```
oClient->RespondPublicMethod(RequestId, Status, "Your Response", azuIoTQoS1);
```

2. デバイスを特定の望ましい状態にするための長期実行コマンドに対する**ツインの望ましいプロパティ**。たとえば、テレメトリの送信間隔を 30 分に設定します。次のメソッドを使用してプロパティを取得できます。

```
oClient->Get_DeviceTwinsProperties("1");
```

3. **クラウドからデバイスへのメッセージ**はデバイスアプリへの一方向通知です。メッセージを受け取るには、まずサブスクライブする必要があります。

```
oClient->Subscribe_CloudToDevice;
```

メッセージは **OnMQTTPublish** イベント（テキストペイロード）または **OnMQTTPublishEx** イベント（Value、Bytes、Stream プロパティを持つ TsgcWSMQTTPublishData としてのペイロード）で受信されます。

```
void AzureIoTMQTTPublish(TsgcWSConnection *Connection, string aTopic, string aText)
{
    DoLog("Received Message: " + aTopic + " " + aText);
}
```

ファイルをアップロードする

IoT ハブは、共有アクセス署名（SAS）URI または X509 証明書を提供することで、接続されたデバイスからのファイルアップロードを容易にします。

SASを選択した場合は、以下のプロパティを設定する必要があります。

- **Azure.IoTHub:** 例 youriothub.azure-devices.net
- **Azure.Deviceld:** 例: myDevice
- **SAS.SecretKey:** 例: Yj7RRPnkSDTv+UCFLgwIP/FrbDymZv4qVAIoTLHUFR8=
- **SAS.KeyName:** 例: iothubowner
- **SAS.Enabled:** 値を true に設定する必要があります。

X509 証明書を選択した場合は、次のプロパティを設定する必要があります:

- **Certificate.CertFile:** PEM証明書へのパスです。
- **Certificate.KeyFile:** PEM キー ファイルへのパス。
- **Certificate.Enabled:** 値を true に設定する必要があります。

UploadFile メソッドを使用して Azure サーバーにファイルをアップロードします。Overwrite パラメータが true に設定されている場合は既存のファイルを置き換えます。Overwrite パラメータが false に設定されている場合、ファイルが存在しない場合のみアップロードし、存在する場合はエラーを発生させます（これがデフォルトのオプションです）。

```
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 IoT では、DPS を使用してコードからデバイスを登録できます。現在、このライブラリはスコープ ID と登録 ID をパラメーターとして渡すことでデバイスを登録する方法をサポートしています。

```

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

Azure IoT Explorer アプリケーションを使用して、IoT Hub に接続されたデバイスと対話できます。受信したテレメトリメッセージ、登録されたデバイスなどを確認できます。このアプリケーションは無料でダウンロードできます。

<https://github.com/Azure/azure-iot-explorer/releases>

HTTP

HTTPプロトコルを使用すると、画像やHTMLドキュメントなどのリソースをサーバーから取得できます。これはクライアントサーバープロトコルであり、クライアントが必要なリソースをサーバーにリクエストすることを意味します。

クライアントがサーバーに接続しようとする、次のステップが実行されます：

1. 新しい TCP 接続を開きます
2. リクエストしたデータをサーバーにメッセージとして送信します。

```
GET / HTTP/1.1
Host: server.com
Accept-Language: en-us
```

3. サーバーから送信されたレスポンスを読み取る

```
HTTP/1.1 200 OK
Server: Apache
Content-Length: 120
Content-Type: text/html
...
```

コンポーネント

- **HTTP/2:** HTTP/2（またはh2）は、プッシュ、ストリームの多重化、フレーム制御をWebにもたらすバイナリプロトコルです。
- **HTTP/1 クライアント:** TIdHTTP クライアントコンポーネントから継承する非ビジュアルコンポーネント。
- **OAuth2:** OAuth2 はサードパーティアプリケーションが HTTP サービスへの制限されたアクセスを受け取ることを可能にします。
- **JWT:** JWTは、いくつかのクレームを主張するJSONペイロードを持つ、オプションの署名および/または暗号化付きデータを作成できます。
- **Amazon SQS:** マイクロサービス、分散システム、サーバーレスアプリケーション向けのフルマネージドメッセージキューイングサービス。
- **Google Cloud Pub/Sub:** アプリケーション間のメッセージングを提供し、アプリケーション間の信頼性の高い多対多の非同期メッセージングを提供するように設計されています。
- **Google カレンダー:** Google Calendar API V3 を使用できます：カレンダーの取得、イベントの取得、独自のカレンダーとの同期など。
- **Google FCM:** Firebase Cloud Messaging を使用して通知を送信します。

HTTP/2

HTTP/2 は HTTP 1.1 プロトコルの進化版であり、基本的にネットワーク使用をより効率的にしようとするものです。セマンティクスは同じであり、旧プロトコルとの互換性を持つように設計されています。

HTTP 1.1 Limitations

HTTP 1.1は1接続あたり1リクエストの処理に制限されているため、通常 クライアントはサーバーからファイルをリクエストするために複数の接続を使用します。しかし これは問題を引き起こします。オープンなTCP接続が多すぎると、サーバーリソースを使用するためのクライアント間の競合が発生し、より多くのクライアントがサーバーに接続するにつれてパフォーマンスがどんどん低下するからです。

主な機能

- HTTP/2 はバイナリプロトコルです（HTTP 1.1 はテキストプロトコルであることを思い出してください）。
- HTTP/2 は TLS と ALPN を使用して動作します。
- 多重化されています（単一の TCP 接続で複数のリクエストを送信できます）。
- サーバーはクライアントにレスポンスをプッシュできます。
- ラウンドトリップタイムを削減し、クライアントがより速く読み込めるようにします。

HTTP/2 には他の改善点もあります。詳細：[RFC7540](#)。

HTTP/2 は ALPN プロトコルが必要なため、カスタム Indy バージョンが必要です。

コンポーネント

- **TsgcHTTP2Client**: HTTP/2 プロトコルを完全にサポートするクライアントコンポーネント（sgcWebSockets 100% Pascal コード、外部ライブラリなし）。
- **TsgcWebsocketHTTPServer**: HTTP/2 プロトコルを完全にサポートするサーバーコンポーネントです（sgcWebSockets 100% Pascal コード、外部ライブラリなし）。デフォルトでは HTTP/2 は無効になっており、HTTPOptions プロパティを使用して Enable = true を設定することで有効にできます。
- **TsgcWebsocketServer_HTTPAPI**: HTTP/2 プロトコルをサポートするサーバーコンポーネント（Microsoft 実装で Windows 2016 以降または Windows 10 以降が必要）。
- **DataSnap サーバー**: DataSnap サーバーも HTTP/2 プロトコルをサポートできます。

API

- **Apple Push Notifications**: サーバープロバイダーからユーザーのデバイスにユーザー向け通知をプッシュします。

TsgcHTTP2Client

TsgcHTTP2Client はクライアント HTTP/2 コンポーネントを実装し、HTTP/2 サーバーに接続できます。

はじめに

TsgcHTTP2Client はクライアントHTTP/2コンポーネントを実装し、HTTP/2サーバーに接続できます。このコンポーネントを設定するには、以下の手順に従ってください:

1. **TsgcHTTP2Client** コンポーネントの新しいインスタンスを作成します。
2. サーバーにリクエストを送信し、OnHTTP2Responseイベントを使用してレスポンスを処理します。使用例:

```
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);
}
```

リファレンス

- プロパティ — コンポーネントで利用可能なプロパティ 19 件。
- メソッド — パブリックメソッド 19 個。
- イベント — 処理可能な13のイベント。

ガイド

- リクエスト
 - [HTTP/2 メソッドリクエスト](#)
 - [HTTP/2 サーバープッシュ](#)
 - [ファイルダウンロード](#)
 - [HTTP/2 部分レスポンス](#)
 - [HTTP/2 ヘッダー](#)
- 接続
 - [クライアント接続のクローズ](#)
 - [クライアント接続のアクティブ維持](#)
 - [HTTP/2 切断理由](#)
 - [クライアントの保留中のリクエスト](#)
- 認証
 - [クライアント認証](#)
 - [HTTP/2 と OAuth2](#)
- クラス
 - [TsgcHTTP2ConnectionClient](#)
 - [TsgcHTTP2RequestProperty](#)

- [TsgcHTTP2ResponseProperty](#)

TsgcHTTP2Client | HTTP/2 メソッドのリクエスト

HTTP/2 クライアントはブロッキングおよびノンブロッキングモードで動作できます。内部的にコンポーネントはセカンダリスレッドで動作し、リクエストは非同期に処理されますが、リクエストを呼び出してそのリクエストが完了するまで待機することもできます。

以下に、クライアントがHTTP/2サーバーにHTMLページを要求する方法と、両方のモードで動作させる方法の例を示します。

非同期モード

<https://www.google.com> の以下のURLを取得し、クライアントが完全なレスポンスを受信したときに通知を受け取ります。**GETASYNC** メソッドを呼び出した後、プロセスは継続し、レスポンスを受信されるとOnHTTP2Response イベントが呼び出されます。

```
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);
}
```

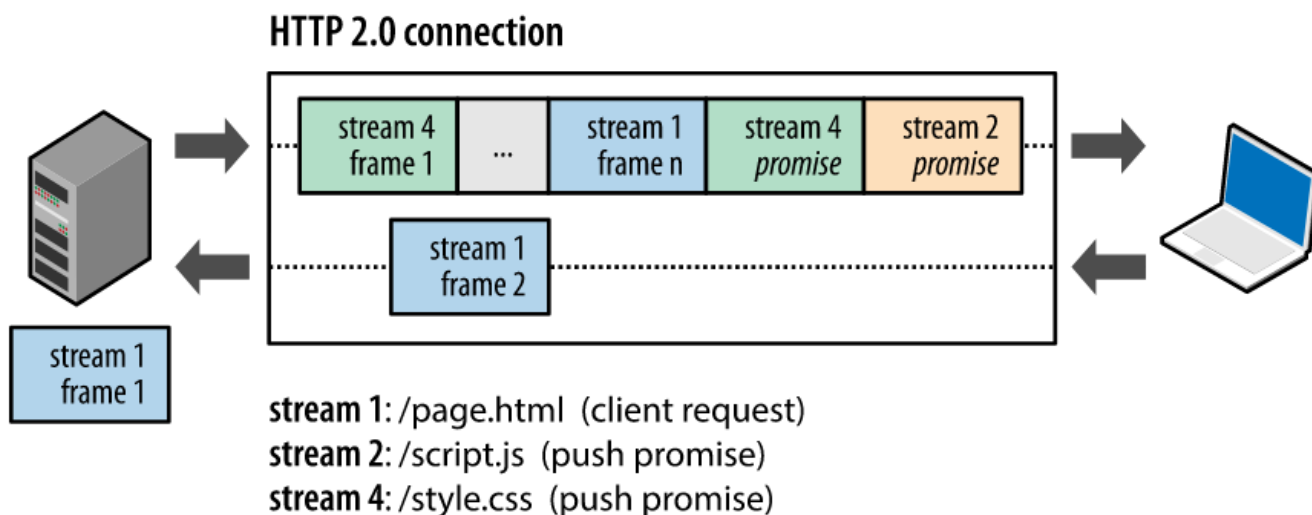
ブロッキングモード

URL <https://www.google.com> を取得し、クライアントが完全な応答を受信するまで待機します。**GET** メソッドを呼び出した後、応答を受信するかタイムアウトになるまでプロセスが待機します。HTTP/2 クライアントの Response プロパティを使用して生のレスポンスデータにアクセスできます。ここでは生のヘッダー、ステータスレスポンスコード、文字セットなどにアクセスできます。

```
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));
```

リクエスト | HTTP/2 サーバープッシュ

サーバープッシュは、単一のクライアントリクエストに対してサーバーが複数のレスポンスを送信する機能です。つまり、元のリクエストへのレスポンスに加えて、サーバーはクライアントが明示的にリクエストしなくても追加のリソースをプッシュできます。



サーバーがクライアントにPushPromiseメッセージを送信するたびに、OnHTTP2PushPromise イベントが呼び出されます。クライアントがPushPromiseを受信すると、サーバーが次のパケットでこのリソースを送信することを意味するため、クライアントはそれを受け入れるか拒否できます。

```
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 ファイルダウンロード

クライアントがサーバーからファイルをリクエストする場合、OnHTTP2Response イベントを使用してストリームレスポンスを読み込みます。

大容量ファイルのダウンロード

大きなファイル（数百 MB 以上）をダウンロードする場合は、デフォルトの 60 秒タイムアウトで転送が中断されないよう、**HTTP2Options.ReadTimeout** を **0**（タイムアウトなし）に設定してください：

```
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 部分レスポンス

通常 HTTP リクエストを送信するとサーバーはリクエストされたファイルの応答を送信しますが、単一の応答の代わりにサーバーがストリームのよう複数の応答を送信する場合があります。その場合は **OnHTTP2ResponseFragment** イベントを使用してこれらの応答をキャプチャしてユーザーに表示できます。

使用例： <https://http2.golang.org/clockstream> にリクエストを送信すると、サーバーは 1 秒ごとにストリームレスポンスを送信します。

```
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 ヘッダー

TsgcHTTP2Client では、クライアントが接続する際にサーバーに送信するヘッダーのカスタマイズが可能です。
使用例: HTTP ヘッダー "Client: sgcWebSockets" を追加する必要がある場合

```
void OnHTTP2BeforeRequest(TObject *Sender, const TsgcHTTP2ConnectionClient *Connection,  
    ref TStringList *Headers)  
{  
    Headers->Add("Client: sgcWebSockets");  
}
```

Request.CustomHeadersを使用して、カスタマイズしたヘッダーを追加することもできます。

TsgcHTTP2Client | クライアント接続のクローズ

接続は Active プロパティを使用するか、Close/Disconnect メソッドを使用して閉じることができます。

Activeプロパティ

接続がアクティブな状態で `Active := False` を設定すると、接続に関する切断メッセージをサーバーに送信せずに即座に接続が閉じられます。

切断

ソケットを切断するには `TsgcHTTP2Client` または `TsgcHTTP2ConnectionClient` の `Disconnect` メソッドを使用できます。

閉じる

このメソッドは、接続が閉じられることをサーバーに通知するメッセージを送信し、オプションで切断の理由に関する情報を送信できます。HTTP/2 接続をクリーンに閉じる方法です。

`Close` メソッドは `TsgcHTTP2Client` または `TsgcHTTP2ConnectionClient` オブジェクトから呼び出すことができます。

以下のエラー理由が送信される可能性があります。

- エラーなし
- protocol error
- 内部エラー
- フロー制御エラー
- settings timeout
- stream closed
- フレームサイズエラー
- refused stream
- cancel
- compression error
- 接続エラー
- enhance your calm
- セキュリティが不十分
- required

TsgcHTTP2Client | Client Keep Connection Active

クライアントがサーバーに接続した後、不十分な信号や接続エラーなどにより接続が閉じられることがあります。接続をアクティブに保つのに役立つ2つのプロパティがあります。

HeartBeat

HeartBeat プロパティにより、**接続を維持**するために X 秒ごとに **Ping を送信**できます。一部のサーバーは、ピア間でデータが交換されない場合に TCP 接続を閉じます。HeartBeat はこの問題を解決し、特定の間隔で ping を送信します。通常、これだけで接続をアクティブに維持するのに十分です。

HeartBeatType プロパティを使用して HeartBeat の動作をカスタマイズできます。

1. **hbtAlways:** Interval で定義された x 秒ごとに ping を送信します。
2. **hbtOnlyIfNoMsgRcvInterval:** Interval プロパティで定義された直近の x 秒間にメッセージが受信されなかった場合にのみ、x 秒ごとに ping を送信します。

使用例： 30秒ごとにピングを送信します。

```
oClient = new TsgcHTTP2Client();
oClient->HeartBeat->Interval = 30;
oClient->HeartBeat->Enabled = true;
oClient->Active = true;
```

WatchDog

WatchDog が有効な場合、クライアントが切断を検出すると、WatchDog は接続が再びアクティブになるまで X 秒ごとに再接続を試みます。

使用例: 切断後に無制限の試行回数で 10 秒ごとに再接続します。

```
oClient = new TsgcHTTP2Client();
oClient->WatchDog->Interval = 10;
oClient->WatchDog->Attempts = 0;
oClient->WatchDog->Enabled = true;
oClient->Active = true;
```

TsgcHTTP2Client | HTTP/2 切断理由

HTTP/2 サーバーはさまざまな理由でクライアントを切断することがあります。サーバーがクライアントに切断の理由を通知する場合は、切断に関する情報を含む **GoAway** メッセージをクライアントに送信します。

OnHTTP2GoAway イベントを使用して、サーバーが切断した理由をキャッチします（クライアントが接続を閉じたい場合は、close メソッドを使用して閉じる理由を送信できます）。

TsgcHTTP2GoAwayProperty オブジェクトには切断に関する情報が含まれています。

- **LastStreamId:** サーバーによって処理された最後のストリームです。
- **ErrorCode:** エラーコードを識別する整数。
- **ErrorDescription:** エラーの説明。次のいずれか:
 - エラーなし
 - protocol error
 - 内部エラー
 - フロー制御エラー
 - settings timeout
 - stream closed
 - フレームサイズエラー
 - refused stream
 - cancel
 - compression error
 - 接続エラー
 - enhance your calm
 - セキュリティが不十分
 - required
- **AdditionalDebugData:** 切断に関する詳細情報を提供するオプションの文字列。

TsgcHTTP2Client | クライアント保留中のリクエスト

クライアントが複数のリクエストを送信する場合、それらはセカンダリスレッドで処理されます。接続が何らかの理由で閉じられた場合でも、保留中のリクエストが残っている場合があります。OnHTTP2PendingRequests イベントを使用してこれらの保留リクエストを処理します。このイベントはクライアントが切断を検出し、保留中のリクエストがまだある場合に呼び出されます。このイベントには2つのパラメータがあります。

1. **Reconnect:** デフォルトでは無効です。True に設定するとクライアントが自動的に再接続します。
2. **Clear:** デフォルトで有効です。False に設定すると、クライアントが再接続したときにサーバーへの保留中のリクエストを再送信しようとします。

TsgcHTTP2Client | クライアント認証

HTTP/2 クライアントは 2 種類の認証タイプをサポートしています: Basic 認証と OAuth2 認証。

両方の認証タイプを処理するには、**OnHTTP2Authorization** イベントを使用します。

基本認証

サーバーが基本認証を要求するヘッダーを返した場合、OnHTTP2Authorization でユーザー名とパスワードを設定します。

```
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

サーバーが Bearer Token Authentication を要求するヘッダーを返した場合は、OnHTTP2Authorization にトークンを設定します。

```
TsgcHTTP2Client *oClient = new TsgcHTTP2Client();
oClient->OnHTTP2Authorization = OnHTTP2AuthorizationEvent;
...
void OnHTTP2AuthorizationEvent(TObject *Sender, const TsgcHTTP2ConnectionClient *Connection, const string AuthType)
{
    if (AuthType == "Bearer")
    {
        aToken = "bearer token";
    }
}
```

サードパーティからの Bearer 値

Bearer 値を別のメソッドで取得してすでに知っている場合は、次のリクエストプロパティを使用して HTTP ヘッダーとして Bearer 値を渡すことができます。HTTP リクエストメソッドを呼び出す前に設定してください:

```
TsgcHTTP2Client.Request.BearerAuthentication
= true

TsgcHTTP2Client.Request.BearerToken = "< value of the token >"
```

OAuth2

HTTP/2 クライアントで OAuth2 コンポーネントを使用する方法については以下の記事をお読みください。

TsgcHTTP2Client | HTTP/2 と OAuth2

OAuth2 は Google などの多くの企業で使われる一般的な認可方法です。Google サーバーに対して認証を行い、その API を使用する場合は、通常 OAuth2 を使用した認証が必要です。

sgcWebSockets は HTTP/2 クライアント下で OAuth2 をサポートしています。[TsgcHTTP_OAuth2](#) のインスタンスを割り当てる **Authentication.Token.OAuth** というプロパティがあります。

GMail Google API への接続方法

Google API に接続するには、TsgcHTTP_OAuth2 のインスタンスを作成して以下のデータを入力する必要があります：

```
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クライアントコンポーネントに入力した後、TsgcHTTP2Clientの新しいインスタンスを作成し、OAuth2コンポーネントをHTTP/2クライアントに割り当てます。

```
TsgcHTTP2Client1.Authentication.Token.OAuth := TsgcHTTP_OAuth1;
```

最後に、yourname@gmail.com アカウントのメッセージリストを取得するリクエストを実行します。

```
oStream := TStringStream.Create('');
Try
  TsgcHTTP2Client1.Get('https://gmail.googleapis.com/gmail/v1/users/yourname@gmail.com/messages', oStream);
  ShowMessage(oStream.DataString);
Finally
  oStream.Free;
End;
```

TsgcHTTP2ConnectionClient

TsgcHTTP2ConnectionClient はクライアント HTTP/2 接続のラッパーです。クライアントイベントでこのオブジェクトにアクセスできます。

メソッド

- **Ping:** 接続を維持するためにサーバーへ ping を送信します。
- **Close:** 切断理由に関する情報をサーバーに送信します。
- **Disconnect:** サーバーに情報メッセージを送信せずに接続を閉じます。
- **HTTP/2 メソッド:**

GET: GET メソッドは指定されたリソースの表現をリクエストします。GET を使用するリクエストはデータの取得のみを行う必要があります。

HEAD: HEAD メソッドは GET リクエストと同一のレスポンスを要求しますが、レスポンスボディは返しません。

POST: POST メソッドは、指定されたリソースにエンティティを送信するために使用され、多くの場合サーバーの状態変更や副作用を引き起こします。

PUT: PUT メソッドはターゲットリソースの現在のすべての表現をリクエストペイロードで置き換えます。

DELETE: HEAD メソッドは GET リクエストと同一のレスポンスを要求しますが、レスポンスボディはありません。

CONNECT: CONNECTメソッドは、ターゲットリソースによって識別されるサーバーへのトンネルを確立します。

OPTIONS: OPTIONS メソッドはターゲットリソースの通信オプションを説明するために使用されます。

TRACE: TRACE メソッドはターゲットリソースへのパスに沿ってメッセージループバックテストを実行します。

PATCH: PATCHメソッドは、リソースに部分的な変更を適用するために使用されます。

TsgcHTTP2RequestProperty

このオブジェクトは OnHTTP2Response イベントの引数として受信されます。サーバーが送信したレスポンスの元のリクエストを知ることができます。

プロパティ

- **Method** : HTTP/2 メソッド (GET、POST...) を識別します。
- **URL**: リクエストされた URL です。
- **Request**: リクエストのフィールドを含みます。

TsgcHTTP2ResponseProperty

このオブジェクトはOnHTTP2Responseイベントの引数として受信され、サーバーがクライアントに送信したレスポンスを知ることができます。

プロパティ

- **Headers:** サーバーから受信した生のヘッダーのリストを含みます。
- **Data:** リクエストへの応答としてサーバーから送信された生のボディが含まれます。
- **DataStream:** Data を文字列に変換したものです。
- **DataUTF8:** DataをUTF8文字列に変換したものです。
- **PushPromise:** 割り当てられている場合、サーバーからクライアントに送信された PushPromise オブジェクトが含まれます（このレスポンスオブジェクトはクライアントからリクエストされていないことを意味します）。

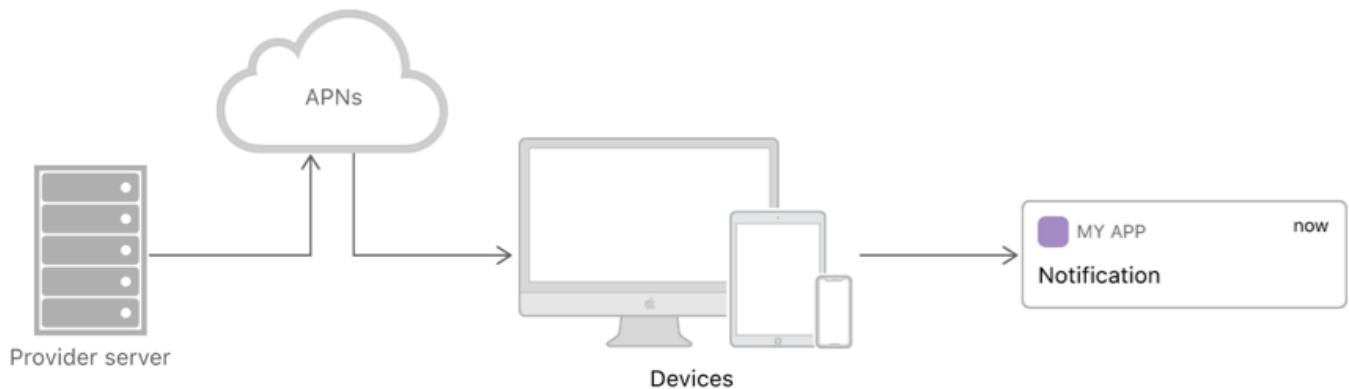
HTTP2 | Apple プッシュ通知

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server

Apple は Apple Push Notification Service (APNs) を使用して Apple デバイスにプッシュ通知を送信することを許可しています。

デバイスに通知を送信する場合、プロバイダーは 以下の情報を含む HTTP/2 POST を APNs に送信する必要があります:

- 送信したい情報を含む JSON ペイロード。
- ユーザーのデバイスを識別する Device Token。
- 通知の配信方法に関するHTTPヘッダー。
- APNs へのリクエストを認証するための SSL 証明書または JWT トークン



通知を送信するために必要なもの

Rad Studio を使用してデバイスに通知を送信するには、次の手順に従う必要があります。

- APNs でアプリを登録する
- [リモート通知の生成](#)
- [APNsへの通知リクエストの送信](#)
 - [APNs へのトークンベース接続](#)
 - [APNs への証明書ベースの接続](#)

APN | リモート通知 APNs の生成

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/generating_a_remote_notification

Apple Notifications は JSON ペイロードを使用して通知オブジェクトを送信します。ペイロードの最大サイズは 4096 バイトです。

JSON ペイロードのサンプル

シンプルなアラートメッセージ

```
{
  "aps": {
    "alert": "Alert from sgcWebSockets!"
  }
}
```

タイトルとサブタイトル付きのアラート。

```
{
  "aps" : {
    "alert" : {
      "title" : "Game Request",
      "subtitle" : "Five Card Draw",
      "body" : "Bob wants to play poker",
    },
    "category" : "GAME_INVITATION"
  },
  "gameID" : "12345678"
}
```

APN | APNs への通知リクエストの送信

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server

リモート通知ペイロードとデバイストークン情報を Apple Push Notification service (APNs) に送信します。

APNs への接続方法

サーバープロバイダーと以下のいずれかのサーバーとの間で接続を確立するには、HTTP/2 プロトコルと、少なくとも TLS 1.2 以降を使用する必要があります：

開発サーバー: <https://api.sandbox.push.apple>

Production Server: <https://api.push.apple>

サンプルコード

TsgcHTTP2Client の新しいインスタンスを作成し、POST メソッドを呼び出して APNs に通知を送信します。

```
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();
}
```

通知を送信するには、HTTP/2 プロトコルと TLS 1.2 以降を使用して APNs とのトークンベースまたは証明書ベースの信頼を確立する必要があります。

- [APNs へのトークンベース接続](#)
- [APNs への証明書ベースの接続](#)

APNs 信頼 | APNs へのトークンベース接続

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/establishing_a_token-based_connection_to_apns

ステートレス認証トークンを使用して、Apple Push Notification service (APNs) との通信をセキュアにします。

まず Apple Developer アカウントから **Encryption Key** と **Key ID** を取得する必要があります。登録が成功すると、キー ID を含む 10 文字の文字列と .p8 拡張子の認証トークン署名キーが得られます。

ES256 をアルゴリズムとして使用して JWT を生成するために sgcWebSockets [JWT クライアント](#)を使用する必要があります。トークンはすべての HTTP/2 リクエストごとに生成してはいけません。トークンは 20 分前に更新してはいけませんし、60 分後に更新してはいけません。

JWTクライアントの設定

次の値でJWTクライアントを設定します:

- **JWTOptions.Header.Algorithm:** はトークンの暗号化に使用した暗号化アルゴリズムです。APNsは ES256 アルゴリズムのみをサポートします。
- **JWTOptions.Header.kid:** 開発者アカウントから取得した10文字のキー ID です。
- **JWTOptions.Payload.iss:** 会社のアプリ開発に使用する 10 文字のチーム ID の値。この値はデベロッパーアカウントから取得してください。
- **JWTOptions.Payload.iat:** この JSON トークンが生成された時刻を示す「発行時刻」の値です。エポックからの秒数を UTC で指定してください。値は現在時刻から 1 時間以内でなければなりません。
- **JWTOptions.RefreshTokenAfter:** 値を 40 分 (60*40) 秒に設定します。

トークンベースの接続を使用するには、アプリのバンドル ID/アプリ ID の値 (例: com.example.application) とともに **apns-topic** を送信する必要があります。

```
TsgcHTTP2Client *oHTTP = new TsgcHTTP2Client(NULL);
oHTTP->TLSOptions->IOHandler = iohOpenSSL;

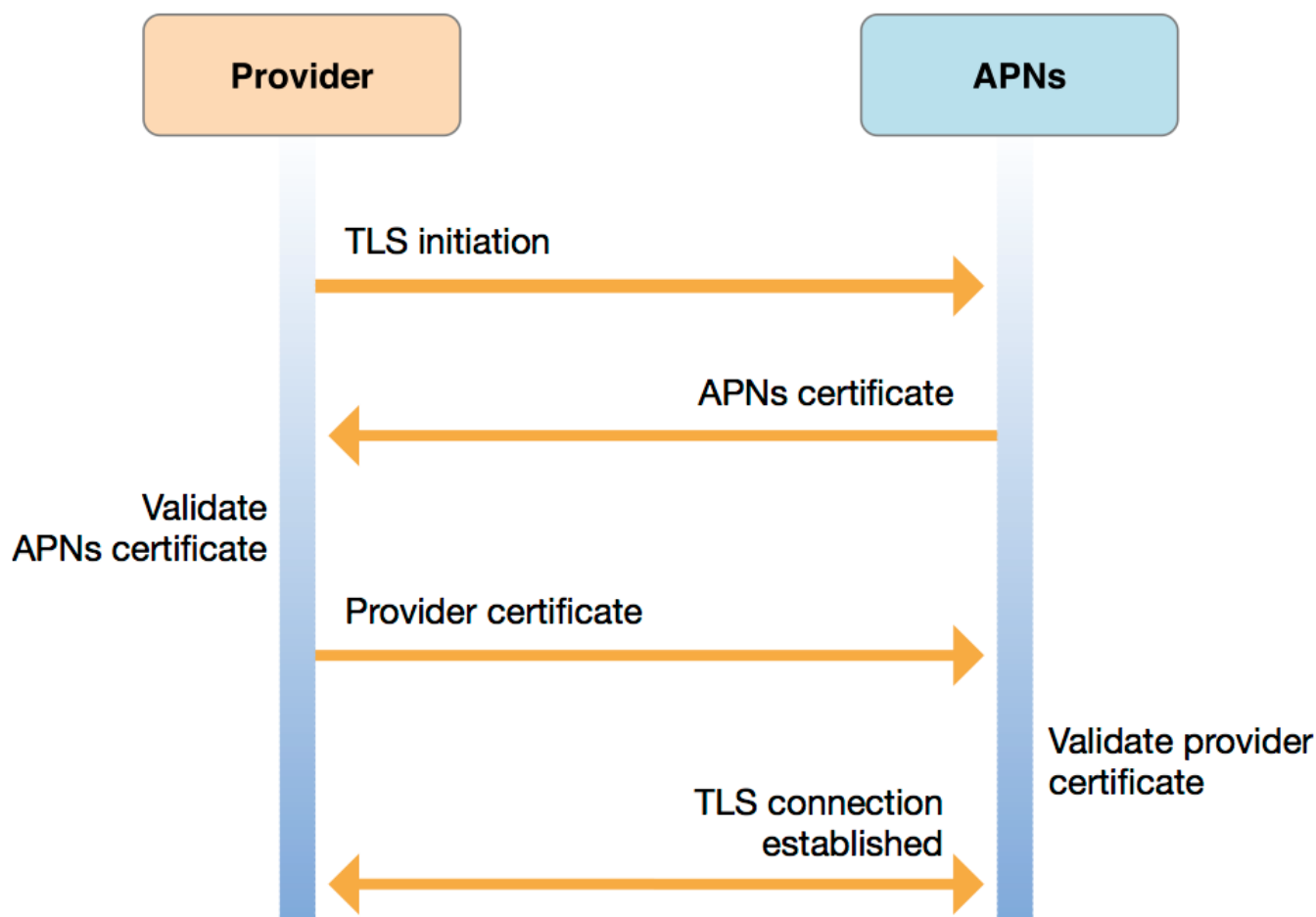
TsgcHTTP_JWT_Client *oJWT = new TsgcHTTP_JWT_Client(NULL);
oHTTP->Authentication->Token->JWT = oJWT;
oJWT->JWTOptions->Header->alg = jwtES256;
oJWT->JWTOptions->Header->kid = "apple key id";
oJWT->JWTOptions->Payload->iss = "issuer";
oJWT->JWTOptions->Payload->iat = StrToInt64(GetDateTimeUnix(Now, False));
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");
```

APNs への証明書ベースの接続

https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/establishing_a_certificate-based_connection_to_apns

Apple Push Notification service (APNs) との通信を、Apple から取得した証明書を使用してセキュリティ保護できます。



まずデベロッパーアカウントにログインして、Apple Push Notification サービス用の**新しい証明書**を作成します

証明書をダウンロードすると、sgcWebSockets HTTP/2 クライアントで 2 つのセキュリティ IOHandler を使用できます（Windows のみ対応。その他のプラットフォームでは OpenSSL のみサポートしています）。

- OpenSSL
- SChannel（Windows のみ）

OpenSSL

OpenSSLを使用する場合、アプリケーションとともにOpenSSLライブラリをデプロイする必要があります。[TsgcHTTP2Client](#)で証明書を設定する前に、OpenSSLはP12証明書を直接インポートできないため、まずPEM形式に変換する必要があります。

以下のコマンドを使用して、単一のP12証明書をPEM形式の証明書と秘密鍵ファイルに変換します

PEM 証明書ファイルの作成

```
openssl pkcs12 -in INFILE.p12 -out OUTFILE.crt -nokeys
```

秘密鍵ファイルを作成する

```
openssl pkcs12 -in INFILE.p12 -out OUTFILE.key -nodes -nocerts
```

PEM 形式の証明書とプライベートキーを取得したら、次のように [TsgcHTTP2Client](#) を設定できます。

```
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

SChannel を使用する場合、ライブラリをデプロイする必要はなく、Apple からダウンロードした証明書を PEM 形式への変換なしに直接インポートできます。

IOHandler として SChannel を使用する場合は、プロパティ UseLegacyCredentials を true に設定します。

```
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;
```

エラー

エラー **"missing topic"** が発生した場合、プッシュ通知、VOIP など複数の用途に使用できるユニバーサル証明書 (証明書) を使用している可能性が高く、アプリのバンドル ID/アプリ ID の値でトピック名を設定する必要があります (例: com.example.application)。HTTP/2 クライアントの Request プロパティで apns-topic ヘッダーに正しい値を設定してください。

```
oHTTP->Request->CustomHeaders->Clear();  
oHTTP->Request->CustomHeaders->Add("apns-topic: com.example.application");
```


HTTP/1

TsgcHTTP1Client は TldHTTP Indy コンポーネントを継承し、いくつかの新しいプロパティを追加した非ビジュアルコンポーネントです。
このコンポーネントはsgcHTTPユニットにあります。

TLSOptions

HTTP/1 プロトコルを使用してセキュアな SSL/TLS サーバーに接続する方法を設定できます。

ALPNProtocols: サーバーに送信されるALPNプロトコルのリスト。

RootCertFile: ルート証明書ファイルへのパス。

CertFile: 証明書ファイルへのパス。

KeyFile: 証明書鍵ファイルへのパス。

Password: 証明書がパスワードで保護されている場合は、ここに設定します。

VerifyCertificate: 証明書を検証する場合はこのプロパティを有効にします。SSL 検証のカスタマイズには **OnSSLVerifyPeer** イベントを使用します。

VerifyDepth: X.509 証明書の検証実行時に許可されるリンクの最大数を表す整数プロパティ。

Version: デフォルトでは TLS 1.0 を使用します。サーバーがより高いTLSバージョンを要求する場合は、ここで選択できます。

プロキシ: プロキシサーバーを経由して接続するかどうかを定義できます。以下のプロキシサーバーに接続できます。

pxyHTTP: HTTPプロキシサーバー。

pxySocks4: SOCKS4 プロキシサーバー。

pxySocks4A: SOCKS4A プロキシサーバー。

pxySocks5: SOCKS5 プロキシサーバー。

IOHandler: TLS を使用した接続に使用するライブラリを選択します。

iohOpenSSL: OpenSSL ライブラリを使用し、Indy コンポーネントのデフォルトです。win32/win64 用の OpenSSL ライブラリのデプロイが必要です。

iohSChannel: Microsoft が Windows 向けに実装したセキュリティプロトコルである Secure Channel を使用します。OpenSSL ライブラリのデプロイが不要です。Windows 32/64 ビットのみで動作します。

OpenSSL_Options: OpenSSL ライブラリの設定。

APIVersion: 使用する OpenSSL API を定義できます。

oslAPI_1_0: API 1.0 OpenSSL を使用します。Indy で最後にサポートされたバージョンです。

oslAPI_1_1: API 1.1のOpenSSLを使用します。カスタムIndyライブラリが必要で、OpenSSL 1.1.1ライブラリ (TLS 1.3サポート付き) の使用が可能になります。

oslAPI_3_0: API 3.0 OpenSSLを使用します。当社のカスタムIndyライブラリが必要であり、OpenSSL 3.0.0ライブラリの使用を可能にします(TLS 1.3サポート付き)。

LibPath: OpenSSL ライブラリの場所を設定できます。

oslpNone: これがデフォルトです。OpenSSL ライブラリはバイナリと同じフォルダーまたは既知のパスに存在する必要があります。

oslpDefaultFolder: すべての IDE パーソナリティに対して、ライブラリが格納されるべき OpenSSL パスを自動的に設定します。

oslpCustomFolder: このオプションが選択されている場合、プロパティ LibPathCustom にフルパスを定義します。

LibPathCustom: LibPath = oslpCustomFolder の場合、ここに OpenSSL ライブラリのフルパスを定義します。

UnixSymLinks: Unix システムでシンボリックリンクの読み込みを有効または無効にします (デフォルトでは有効です。OSX64 は除きます)。

oslsSymLinksDefault: デフォルトでは、OSX64 以外でシンボリックリンクが有効です (macOS Monterey 以降では、バージョンなしでライブラリをロードしようとするとう失敗します)。

oslsSymLinksLoadFirst: バージョン管理されたライブラリを読み込む前に、まずシンボリックリンクを読み込みます。

oslsSymLinksLoad: バージョン付きライブラリの読み込みを試みた後にシンボリックリンクを読み込みます。

oslsSymLinksDontLoad: SymLinksを読み込みません。

MinVersion: ここに、クライアントがセキュアなサーバーに接続するために使用する最小バージョンを設定します。デフォルトでは、値はtlsUndefinedで、これは最小バージョンがVersionプロパティで

設定されたものと同じであることを意味します。例: クライアントがTLS 1.2またはTLS 1.3のみを使用して接続するように設定したい場合は、次の値を設定します。

`SSLOptions.Version := tls1_3;`
`SSLOptions.OpenSSL_Options.MinVersion := tls1_2;`
X509Checks: 追加の X509 証明書検証を有効にするにはこのプロパティを使用してください:
Mode: 検証するオプションを選択します。
oslx509chHostName: ホスト名証明書を検証します。
oslx509chIPAddress: 証明書の IP アドレスを検証します。
HostName: リクエストと異なる場合はホスト名を設定します。
IPAddress: リクエストと異なる場合は IP アドレスを設定します。

SChannel_Options: Windows 証明書ストアから証明書を使用できます。
CertHash: 証明書のハッシュ値です。PowerShell で `dir` コマンドを実行することで証明書のハッシュ値を確認できます。
CipherList: 使用する暗号スイートを (":" で区切って) 設定できます。例:
`CALG_AES_256:CALG_AES_128`
CertStoreName: 証明書が保存されているストア名。以下のいずれかを選択してください。
`scsnMY` (デフォルト)
`scsnCA`
`scsnRoot`
`scsnTrust`
CertStorePath: 証明書が保存されているストアパス。以下のいずれかを選択してください:
`scspStoreCurrentUser` (デフォルト)
`scspStoreLocalMachine`

ログ

Log プロパティが有効な場合、ソケットメッセージを指定されたログファイルに保存します。デバッグに役立ちます。

LogOptions.FileName: ファイル名のフルパス。

認証

[OAuth2](#) または [JWT](#) を使用して認証できます。

非同期リクエスト

デフォルトでは、HTTP1ClientはブロッキングリクエストをするためHTTPリクエストメソッドを呼び出した後、クライアントはサーバーからのレスポンスを待ちます。あるいは、非同期メソッドを使用してこれらのHTTPリクエストを別スレッドで実行し、リクエストが呼び出されたスレッドをブロックしないようにすることができます。次の非同期メソッドが実装されています:

- `GetAsync`
- `PostAsync`
- `PutAsync`
- `OptionsAsync`
- `DeleteAsync`

これらのメソッドを呼び出した後、レスポンスを待機せずにコードは次の行に進み、レスポンスはイベント **OnAsyncResponse** を使用して処理できます。

```
void __fastcall OnAsyncResultEvent(TObject* Sender, const TsgcHTTPAsyncRequest* aRequest, const TIdHTTPResponse*
```

非同期リクエストの処理中にエラーが発生した場合、イベント **OnAsyncException** で例外が発生します。

使用例

HTTPs サーバーへ **GET** メソッドをリクエストし、**TLS 1.2** を使用します。

```
TsgcHTTP1Client *oHTTP = new TsgcHTTP1Client();
try
{
    oHTTP->TLSOptions->Version = tls1_2;
    ShowMessage(oHTTP->Get("https://www.google.es"));
}
__finally
{
    oHTTP->Free();
}
```

openssl 1.1 と **TLS 1.3** を使用して HTTPs サーバーに **GET** リクエストを送信

```
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();
}
```

非同期**POST**メソッドを要求し、**OnAsyncResultEvent**を使用して応答を読み取ります。

```
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);
```

Windows 用 **SChannel** を使用して HTTPS サーバーに GET メソッドをリクエストします。

```
TsgcHTTP1Client *oHTTP = new TsgcHTTP1Client();
try
{
    oHTTP->TLSOptions->IOHandler = iohSChannel;
    oHTTP->TLSOptions->Version = tls1_2;
    ShowMessage(oHTTP->Get("https://www.google.es"));
}
__finally
{
    oHTTP->Free();
}
```

データイベントを取得するために **SSE** メソッドをリクエストします

```
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);  
}
```

イベント

OnSSEMessage

新しいSSEメッセージを受信したときにイベントが呼び出されます。

OnSSLVerifyPeer

証明書の検証が有効な場合、このイベントでサーバー証明書を検証して受け入れるかどうかを決定できます。

OnSSLGetHandler

このイベントは SSL ハンドラーが作成される前に発生します。ここで独自の SSL ハンドラーを作成し（TidServerIOHandlerSSLBase または TidIOHandlerSSLBase を継承する必要があります）、必要なプロパティを設定できます。

OnSSLAfterCreateHandler

カスタムSSLオブジェクトが作成されていない場合、OpenSSLハンドラーを使用してデフォルトのオブジェクトが作成されます。SSLハンドラーのプロパティにアクセスして、必要に応じて変更できます。

OnAsyncResult

このイベントは、Asyncメソッド（GetAsync、PutAsyncなどのメソッドを使用）を要求した後に呼び出されます。Responseパラメータを使用して、要求の結果を確認してください。

OnAsyncException

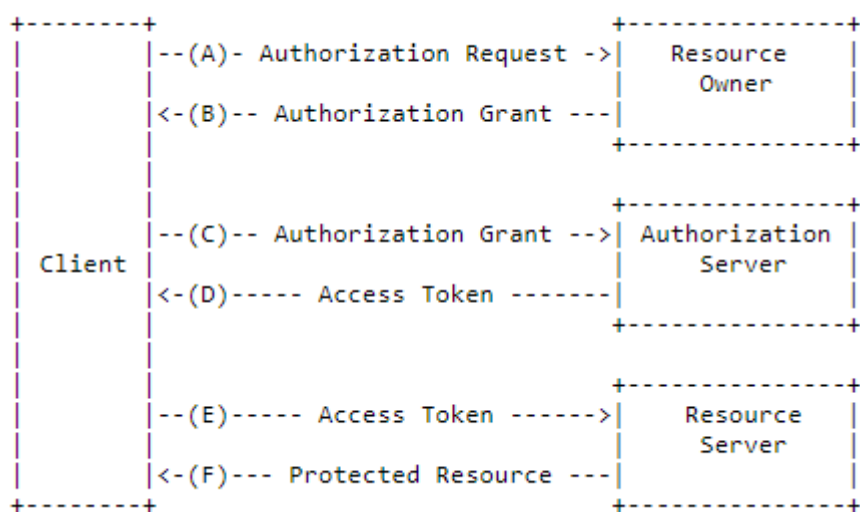
非同期リクエストの処理中にエラーが発生した場合、このイベントは発生した例外とともに呼び出されます。

HTTP | OAuth2

OAuth2はサードパーティのアプリケーションが、リソースオーナーの代わりに、またはサードパーティアプリケーション自身の代わりにアクセスを取得することで、HTTPサービスへの限定的なアクセスを受け取ることを可能にします。OAuth2のおかげで、サービスプロバイダーとコンシューマーアプリケーションは安全な方法で相互にやり取りできます。

OAuth2 には 4 つのロールがあります:

- **Resource Owner:** ユーザー。
- **Resource Server:** 保護されたリソースをホストし、アクセストークンに基づいてそれへのアクセスを提供するサーバー。
- **クライアント:** 許可を求める外部アプリケーション。
- **Authorization Server:** ユーザーを認証した後にアクセストークンを発行します。



コンポーネント

- **TsgcHTTP_OAuth2_Client:** OAuth2をサポートするクライアントであり、GoogleやFacebookなどのOAuth2サーバーに接続して認証を要求できます。
- **TsgcHTTP_OAuth2_Server:** OAuth2 プロトコルのサーバー実装で、サーバーのリソースを保護できます。
- **TsgcHTTP_OAuth2_Server_Provider:** サーバーに外部 OAuth2 プロバイダー（Azure AD、Google、Facebook など）を実装できるため、ユーザーは Azure、Google、Facebook などのユーザー資格情報を使用してログインできます。

サーバーとクライアントのOAuth2コンポーネントは**PKCE**（Proof Key for Code Exchange）をサポートしています。これは、CSRF認可コードインジェクション攻撃を防ぐための認可コードフローの拡張です（RFC 7636）。

TsgcHTTP_OAuth2_Client

クライアントが新しいアクセストークンを必要とするとき、自動的に HTTP サーバーを起動して認可サーバーからのレスポンスを処理します。

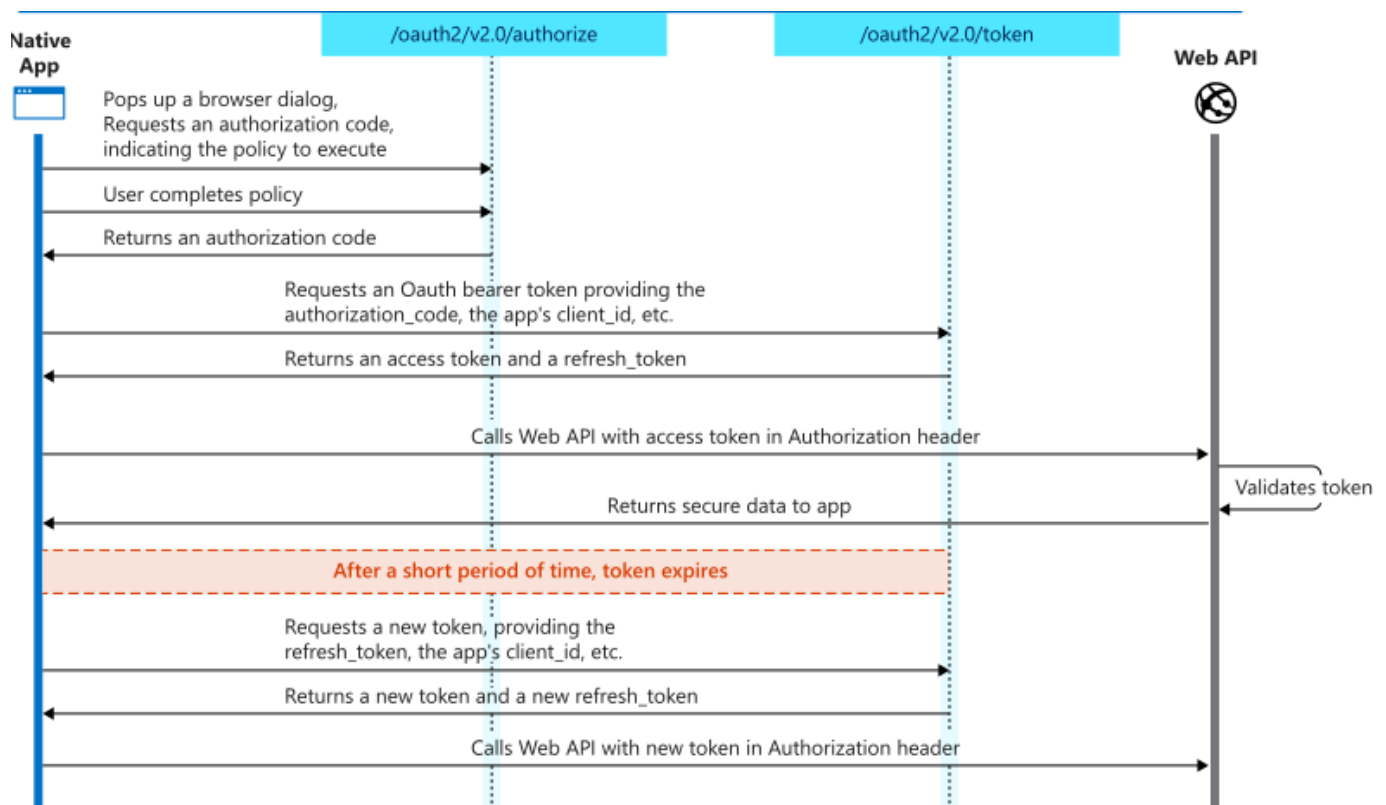
はじめに

このコンポーネントにより、クライアントと他のロール間のフローを処理できます。基本的に `Active := True` を設定すると新しい Web ブラウザーが開き、ユーザーへの認可付与をリクエストします。成功した場合、認可サーバーはアプリケーションにトークンを送信し、このトークンが処理されてクライアントはリソースサーバーに接続できます。このコンポーネントは認可サーバーの応答を処理するシンプルな HTTP サーバーを起動し、アクセストークンをリクエストするための HTTP クライアントを使用します。

GrantType

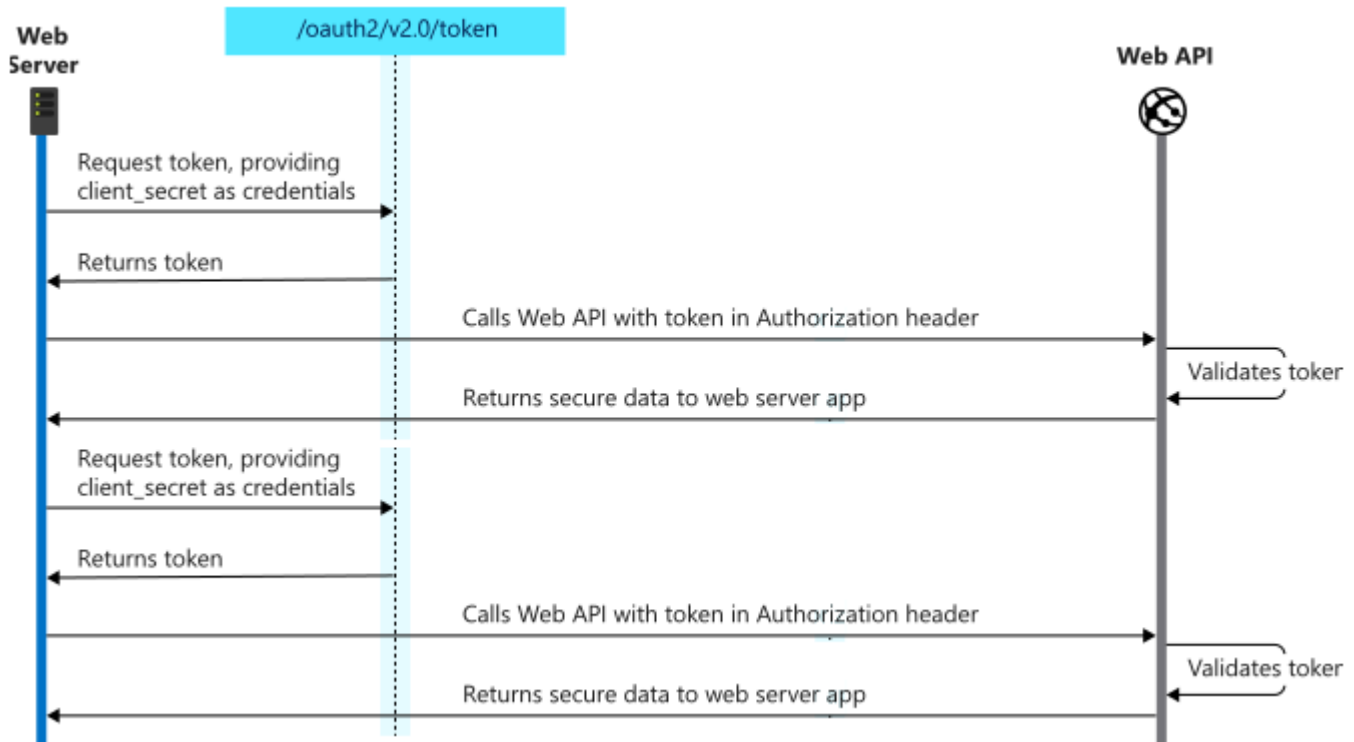
クライアントは以下の種類の認証をサポートします:

auth2Code: シングルページアプリケーション、ウェブアプリケーション、ネイティブインストールアプリケーションを含む大多数のアプリケーションタイプで認証と認可を行うために使用されます。このフローにより、アプリはリソースへのアクセスに使用できる `access_token`、追加の `access_token` を取得するためのリフレッシュトークン、およびサインインユーザーの ID トークンを安全に取得できます。



auth2CodePKCE: PKCE が有効になっている点を除いて `auth2Code` と同じ認証フローです。PKCE (Proof Key for Code Exchange) は OAuth 2.0 のセキュリティ拡張で、ネイティブおよびシングルページアプリケーションの認可フローのセキュリティを強化するために設計されています。特に認可コードが転送中に傍受される可能性があるパブリッククライアントにおいて、傍受攻撃のリスクを軽減します。通常、このオプションはネイティブアプリとモバイルアプリで使用されます。

auth2ClientCredentials: このタイプのグラントは、ユーザーとの即時のやり取りなしにバックグラウンドで実行する必要があるサーバー間のやり取りで一般的に使用されます。このようなアプリケーションは、しばしばデーモンまたはサービスアカウントと呼ばれます。



auth2DeviceCode: ブラウザが使用できない、または入力機能が制限されているスマートテレビ、メディアコンソール、IoT デバイスなどの入力制限デバイス用デバイス認可グラント (RFC 8628)。デバイスにユーザーコードと確認 URI が表示され、ユーザーはセカンダリデバイス (スマートフォンまたはコンピューター) でその URI にアクセスしてコードを入力して認可します。

LocalServerOptions

クライアントが新しいアクセストークンを必要とする場合、認可サーバーからのレスポンスを処理するために自動的に HTTP サーバーを起動します。このサーバーはユーザーには透過的で、通常は localhost で動作します。デフォルトではポート 8080 を使用しますが、必要に応じて変更できます。

- **IP:** サーバーのリスニング IP アドレス、例: 127.0.0.1
- **Port:** デフォルトは 8080。GrantType = auth2CodePKCE (デスクトップおよびネイティブアプリケーション) を使用する場合、ポートの値をゼロに設定するとサーバーがランダムなポートを選択します。
- **RedirectURL:** (オプション) リダイレクト URL のカスタマイズを許可します。例: <http://localhost:8080/oauth/>。
- **SSL:** ローカルサーバーがセキュアなポートで実行される場合は、このプロパティを有効にします (*Professional および Enterprise エディションでのみサポートされます)。
- **SSLOptions:** サーバーの SSL プロパティをカスタマイズできます (*Professional および Enterprise Edition のみサポートされています)。
- **LogOptions:** HTTP 内部サーバーが受信および送信したリクエスト/レスポンスのログを保存できます (*Professional および Enterprise エディションのみ)。
 - **Enabled:** True に設定してファイルへのログを有効にします。
 - **FileName:** ログファイルを保存するファイル名を設定します。

AuthorizationServerOptions

ここでは認可とアクセストークンの URL を設定する必要があります。通常これらは API 仕様で提供されます。Scope はクライアントがリクエストするすべてのスコープのリストです。例:

- **AuthURL:** <https://accounts.google.com/o/oauth2/auth>
- **TokenURL:** <https://accounts.google.com/o/oauth2/token>
- **Scope :** <https://mail.google.com/>
- **RevocationURL:** トークン失効エンドポイントの URL (Revoke メソッドに必須)。例: <https://accounts.google.com/o/oauth2/revoke>
- **IntrospectionURL:** トークンイントロスペクションエンドポイントの URL (Introspect メソッドに必要)。例: <https://accounts.google.com/o/oauth2/introspect>

OAuth2Options

ClientId はサーバーにクライアントの識別情報を知らせる必須フィールドです。ClientId の取得方法については API 仕様を確認してください。クライアントシークレットについても同様です。

サーバーが Basic Authentication を使用したユーザーとパスワードの要求をする場合、Username/Password フィールドで設定できます。例:

- **GrantType:** 認可フローのタイプ
 - **auth2Code:** ウェブサーバーなどの信頼できるアプリ。
 - **auth2CodePKCE:** 信頼できないネイティブまたはモバイルアプリ。
 - **auth2ClientCredentials:** ユーザーインタラクションなしの自動化されたアプリ。
 - **auth2ResourceOwnerPassword:** パスワードを直接処理してユーザーをサインインさせるアプリケーション。
 - **auth2DeviceCode:** ブラウザーまたは入力機能が制限された入力制約のあるデバイス向けのデバイス認可グラント (RFC 8628)。
- **ClientId:** 180803918307-eqjtm20gqfhcs6gjk1brrreng022mqqc.apps.googleusercontent.com
- **ClientSecret:** _by1iYYrvVHxC2Z8TbtNEYJN
- **Username:**
- **Password:**

HTTPClientOptions

ここでは HTTP サーバーに接続して新しいトークンをリクエストする際のクライアントオプションを設定できます。

TLSOptions: TLS が有効な場合、ここでいくつかの TLS プロパティをカスタマイズできます。

ALPNProtocols: サーバーに送信される ALPN プロトコルのリスト。

RootCertFile: ルート証明書ファイルへのパス。

CertFile: 証明書ファイルへのパス。

KeyFile: 証明書鍵ファイルへのパス。

Password: 証明書がパスワードで保護されている場合は、ここに設定します。

VerifyCertificate: 証明書を検証する必要がある場合は、このプロパティを有効にしてください。

VerifyDepth: X.509 証明書の検証実行時に許可されるリンクの最大数を表す整数プロパティ。

Version: デフォルトでは TLS 1.0 を使用します。サーバーがより高い TLS バージョンを要求する場合、ここで選択できます。

IOHandler: TLS を使用した接続に使用するライブラリを選択します。

iohOpenSSL: OpenSSL ライブラリを使用し、Indy コンポーネントのデフォルトです。win32/win64 用の openssl ライブラリをデプロイする必要があります。

iohSChannel: Microsoft が Windows 用に実装したセキュリティプロトコルである Secure Channel を使用します。OpenSSL ライブラリをデプロイする必要がありません。Windows 32/64 ビットでのみ動作します。

OpenSSL_Options: 使用する OpenSSL API を定義できます。

APIVersion: 使用する OpenSSL API を定義できます。

oslAPI_1_0: API 1.0 OpenSSL を使用します。Indy で最後にサポートされたバージョンです。

oslAPI_1_1: API 1.1 の OpenSSL を使用します。カスタム Indy ライブラリが必要で、OpenSSL 1.1.1 ライブラリ (TLS 1.3 サポート付き) の使用が可能になります。

oslAPI_3_0: API 3.0 OpenSSL を使用します。当社のカスタム Indy ライブラリが必要であり、OpenSSL 3.0.0 ライブラリの使用を可能にします (TLS 1.3 サポート付き)。

LibPath: openssl ライブラリの場所を設定できます

oslpNone: これはデフォルトです。openssl ライブラリは、バイナリと同じフォルダー、または既知のパスにある必要があります。

oslpDefaultFolder: すべての IDE パーソナリティに対してライブラリが配置されるべき openssl パスを自動的に設定します。

oslpCustomFolder: このオプションが選択されている場合、プロパティ LibPathCustom にフルパスを定義します。

LibPathCustom: LibPath = oslpCustomFolder の場合、openssl ライブラリが格納されているフォルダーの完全パスをここで定義します。

UnixSymLinks: Unix システムでシンボリックリンクの読み込みを有効または無効にします (デフォルトでは有効です。OSX64 は除きます)。

oslsSymLinksDefault: デフォルトでは有効です。ただし OSX64 では無効です (MacOS Monterey 以降、バージョンなしでライブラリを読み込もうとすると失敗するため)。

oslsSymLinksLoadFirst: SymLinks を読み込み、バージョンライブラリを読み込む前に行います。

oslsSymLinksLoad: バージョンライブラリの読み込みを試みた後にシンボリックリンクを読み込みます。

osIsSymLinksDontLoad: SymLinksを読み込みません。

SChannel_Options: Windows 証明書ストアから証明書を使用できます。

CertHash: 証明書のハッシュ値です。PowerShell で dir コマンドを実行することで証明書のハッシュ値を確認できます。

CertStoreName: 証明書が保存されているストアの名前。以下のいずれかを選択します:

scsnMY (デフォルト)

scsnCA

scsnRoot

scsnTrust

CertStorePath: 証明書が格納されているストアパス。以下から選択してください:

scspStoreCurrentUser (デフォルト)

scspStoreLocalMachine

LogOptions: ファイル名が設定されている場合、HTTP クライアントの HTTP リクエスト/レスポンスのログを保存します。

リファレンス

- プロパティ — コンポーネントで利用可能なプロパティ 6 件。
- メソッド — 7 つのパブリックメソッド。
- イベント — 処理可能な 19 個のイベント。

Web アプリケーション用 OAuth2 クライアント

OAuth2クライアントがWebアプリケーションに接続して認可済みトークンを取得する必要がある場合、ローカルサーバー（認可済みトークンの取得に使用）をWebアプリケーションのパラメータで構成する必要があります。次の**LocalServerOptions** プロパティを設定します。

- **IP:** OAuth2 設定で以前に設定されたアドレス IP。
- **Port:** OAuth2 設定で事前に設定されたリスニングポート。
- **SSL:** サーバーがセキュア接続を使用している場合は、このオプションを有効にします。

デスクトップアプリケーション向け OAuth2 クライアント

OAuth2 クライアントがデスクトップまたはネイティブアプリケーションの場合、承認済みトークンを取得するためのローカルサーバー（LocalServer）はローカルIPアドレス（例: 127.0.0.1）でリッスンでき、ポートはランダムに選択できます。したがって、**LocalServerOptions** は次のように設定する必要があります。

- **IP:** 127.0.0.1
- **Port:** ゼロに設定すると、サーバーは自動的にランダムポートを選択します。

デスクトップまたはネイティブアプリケーション用のOAuth2クライアントを作成する場合、追加のセキュリティを加えるために**OAuth2Options.GrantType**を**auth2CodePKCE** に設定します。

OAuth2 | TsgcHTTP_OAuth2_Client_Google

このコンポーネントを使用すると、Google アカウントに簡単にログインできます。

設定

このモジュールでは、まずGoogleアカウントで **OAuth2アプリケーションを設定する** 必要があります。設定が完了したら、アプリケーションに数行のコードを追加するだけで、ユーザーが任意のGoogleアカウントでログインできるようになります。

Googleからのレスポンスを読み取るために使用される **ローカルサーバー** は、デフォルトで **IPアドレス 127.0.0.1** および **ポート 0**（ランダムポート）でリッスンします。したがって、GoogleアプリケーションでCallBack URLを設定する必要があります。もちろん、IPアドレスとポートを変更することもできます。

GoogleアカウントでOAuth2アプリケーションを設定したら、**TsgcHTTP_OAuth2_Client_Google** のインスタンスを作成し、**Authenticate** メソッドを呼び出して、パラメータとして **Client_Id** と **Client_Secret**を渡すだけです。この **メソッド**は（デフォルトでは最大60秒間）ユーザーが **正常にログイン**するまで待機します。ユーザーが認証されたかどうか、名前、IDなど、ユーザープロフィールのその他のデータを確認できるオブジェクトを返します。

使用例

```
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

このコンポーネントを使用すると、Microsoftアカウントで簡単にログインできます。

設定

このモジュールでは、まずMicrosoftアカウントで **OAuth2アプリケーションを設定する** 必要があります。設定が完了したら、アプリケーションに数行のコードを追加するだけで、ユーザーが任意のMicrosoftアカウントでログインできるようになります。

Microsoftからのレスポンスを読み取るために使用される **ローカルサーバー** は、デフォルトで **IPアドレス 127.0.0.1** および **ポート 8080** でリッスンし、SSLを使用します。したがって、Microsoftアプリケーションで **CallBack URL** を **https://localhost:8080** として設定する必要があります（MicrosoftはローカルIPアドレスとしてlocalhostのみを許可します）。もちろん、IPアドレスとポートを変更することもできます。

MicrosoftアカウントでOAuth2アプリケーションを設定したら、**TsgcHTTP_OAuth2_Client_Microsoft** のインスタンスを作成し、**Authenticate** メソッドを呼び出して、パラメータとして **TenantId**、**Client_Id** と **Client_Secret** を渡すだけです。この **メソッド** は（デフォルトでは最大60秒間）ユーザーが **正常にログイン** するまで待機します。ユーザーが認証されたかどうか、名前、IDなど、ユーザープロファイルのその他のデータを確認できるオブジェクトを返します。

使用例

```
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 | 認可コードグラント (RFC 6749)

概要

Authorization Code グラントは最も一般的な OAuth2 フローです。クライアントシークレットを安全に保存できる Web アプリケーション、デスクトップアプリケーション、モバイルアプリケーション向けに設計されています。このフローでは、ユーザーを認証サーバーにリダイレクトし、そこでユーザーが認証して権限を付与します。その後、サーバーはリダイレクト URI に認証コードを返し、クライアントはそのコードをアクセストークンと交換します。

フロー

1. クライアントは `client_id`、`redirect_uri`、`response_type=code`、`scope`、`state` パラメータとともにユーザーを **AuthURL** にリダイレクトします。
2. ユーザーが認可サーバーで認証し、クライアントに権限を付与します。
3. 認可サーバーは `code` パラメータを含むリダイレクト URI にユーザーをリダイレクトします。
4. クライアントはアクセストークンと引き換えに、`client_id`、`client_secret`、`redirect_uri` とともに認証コードを **TokenURL** に POST します。
5. サーバーは `access_token`、`token_type`、`expires_in`、およびオプションで `refresh_token` を返します。

設定

プロパティ	説明
<code>OAuth2Options.GrantType</code>	auth2Code に設定します。
<code>OAuth2Options.ClientId</code>	認可サーバーが発行したクライアント識別子です。
<code>OAuth2Options.ClientSecret</code>	認証サーバーによって発行されたクライアントシークレット。
<code>AuthorizationServerOptions.AuthURL</code>	ユーザーが認証のためにリダイレクトされる認可エンドポイント URL。
<code>AuthorizationServerOptions.TokenURL</code>	認証コードがアクセストークンに交換されるトークンエンドポイント URL。
<code>AuthorizationServerOptions.Scope</code>	アクセスリクエストのスコープ（例: openid、profile、email）。
<code>LocalServerOptions.IP</code>	ローカルリダイレクトサーバーの IP アドレス（例: 127.0.0.1）。
<code>LocalServerOptions.Port</code>	ローカルリダイレクトサーバーのポート（例: 8080）。

イベント

イベント	説明

<code>OnBeforeAuthorizeCode</code>	クライアントをユーザーが認可エンドポイントにリダイレクトされる前に発生します。認可リクエストのカスタマイズが可能です。
<code>OnAfterAuthorizeCode</code>	サーバーから認可コードが受信された後に発生します。
<code>OnErrorAuthorizeCode</code>	認証コードステップ中にエラーが発生したときに発生します。
<code>OnBeforeAccessToken</code>	クライアントが認可コードをアクセストークンと交換する前に発生します。
<code>OnAfterAccessToken</code>	アクセストークンを受信した後に発生します。このイベントを使用してトークン値を取得してください。
<code>OnErrorAccessToken</code>	トークン交換ステップ中にエラーが発生したときに発生します。

使用例

```
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();
```

トークンのリフレッシュ

アクセストークンの有効期限が切れた場合、リフレッシュトークンを使用してユーザーの操作なしに新しいトークンを取得できます。リフレッシュトークンの値を指定して `Refresh` メソッドを呼び出します。

`OnAfterAccessToken` イベントが新しいアクセストークンとともに発生します。リフレッシュが失敗した場合は `OnErrorAccessToken` イベントが発生します。

使用するタイミング

Authorization Code グラントを使用する場合:

- ・ クライアントシークレットを安全に保存できるサーバーサイドバックエンドを持つ Web アプリケーション。
- ・ サーバーサイドレンダリングアプリケーション（例：ASP.NET、PHP、Java）。
- ・ クライアントシークレットを合理的に保護できるデスクトップアプリケーション。

クライアントシークレットを安全に保存できないパブリッククライアント（ネイティブアプリ、SPA）には、代わりに [PKCE を使用した認証コード](#) の使用を検討してください。

OAuth2 | PKCE を使用した Authorization Code (RFC 7636)

概要

PKCE (Proof Key for Code Exchange) 付き認可コードグラントは、標準的な認可コードフローに追加のセキュリティ層を追加します。クライアントシークレットを安全に保存できないネイティブアプリケーション、モバイルアプリケーション、シングルページアプリケーション (SPA) 向けに設計されています。クライアントシークレットに依存する代わりに、クライアントは認可コードを交換するエンティティが同じエンティティであることを証明する暗号化コード検証子とチャレンジを生成します。

PKCE の仕組み

1. クライアントはランダムな **code_verifier** (暗号学的にランダムな 32 バイト、Base64URL エンコード) を生成します。
 2. クライアントは、**code_challenge** を `code_verifier` の SHA-256 ハッシュ (Base64URL エンコード) として計算します。
 3. クライアントは認可リクエストとともに `code_challenge` と `code_challenge_method=S256` を送信します。
 4. ユーザー認可後、クライアントは元の `code_verifier` をトークン交換要求とともに送信します。
 5. 認可サーバーは、トークンを発行する前に `SHA256(code_verifier) == code_challenge` を検証します。
- これにより、認可コードを傍受した攻撃者が元の `code_verifier` なしにコードを交換できないため、認可コード傍受攻撃を防ぎます。

設定

プロパティ	説明
<code>OAuth2Options.GrantType</code>	auth2CodePKCE に設定します。
<code>OAuth2Options.ClientId</code>	認可サーバーが発行したクライアント識別子です。
<code>OAuth2Options.ClientSecret</code>	オプション。一部のプロバイダーは PKCE を使用しても必要としますが、不要なプロバイダーもあります。
<code>AuthorizationServerOptions.AuthURL</code>	ユーザーが認証のためにリダイレクトされる認可エンドポイント URL。
<code>AuthorizationServerOptions.TokenURL</code>	認証コードがアクセストークンに交換されるトークンエンドポイント URL。
<code>AuthorizationServerOptions.Scope</code>	アクセスリクエストのスコープ。
<code>LocalServerOptions.IP</code>	ローカルリダイレクトサーバーの IP アドレス (例: 127.0.0.1)。
<code>LocalServerOptions.Port</code>	ローカルリダイレクトサーバーのポート。ランダムな利用可能ポートには 0 を設定します。

使用例

```
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();
```

ランダムポート

LocalServerOptions.Port が 0 に設定されている場合、コンポーネントは利用可能なランダムなポートを自動的に選択します。これはポートの競合を避けるためにデスクトップおよびモバイルアプリケーションに推奨されます。選択されたポートは認可サーバーに送信されるリダイレクト URI に含まれます。

トークンのリフレッシュ

PKCEフローは通常リフレッシュトークンを返します。新しいアクセストークンを取得するには Refresh メソッドを使用します:

使用するタイミング

以下の場合に Authorization Code with PKCE グラントを使用します:

- デスクトップアプリケーション (Windows、macOS、Linux のネイティブ)。
- モバイルアプリケーション (iOS、Android) です。
- ブラウザで動作するシングルページアプリケーション (SPA)。
- クライアントシークレットを安全に保存できないパブリッククライアント。
- 機密クライアントの場合でも、標準の [Authorization Code](#) グラントよりセキュアな代替手段として。

OAuth2 | クライアント認証情報グラント (RFC 6749)

概要

Client Credentialsグラントは、ユーザー操作が不要なマシン間 (M2M) 認証に使用されます。クライアントは、独自の資格情報 (client_idとclient_secret) を使用して認可サーバーと直接認証し、アクセストークンを受け取ります。認可コードのステップやユーザーログインはありません。

このグラントタイプは、クライアントがユーザーの代わりではなく自身のために行動する場合に適しています。

フロー

1. クライアントは client_id、client_secret、grant_type=client_credentials、scope をトークンエンドポイントにPOSTします。
2. 認可サーバーがクライアントの資格情報を検証します。
3. サーバーは access_token、token_type、および expires_in を返します。

このグラントタイプではリフレッシュトークンは通常発行されません。クライアントが自分の資格情報を使用して新しいアクセス トークンを要求できるためです。

設定

プロパティ	説明
<code>OAuth2Options.GrantType</code>	auth2ClientCredentials に設定します。
<code>OAuth2Options.ClientId</code>	認可サーバーが発行したクライアント識別子です。
<code>OAuth2Options.ClientSecret</code>	認証サーバーによって発行されたクライアントシークレット。
<code>AuthorizationServerOptions.TokenURL</code>	トークンエンドポイントのURLです。ユーザー認可ステップがないため、AuthURLは不要です。
<code>AuthorizationServerOptions.Scope</code>	アクセスリクエストのスコープ。

使用例

```
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();
```

使用するタイミング

クライアント認証情報グラントを使用するのは以下の場合:

- ユーザーの操作なしに実行されるバックエンドサービスとデーモン。
- サーバー間 API 通信。
- スケジュールされたジョブ、バッチプロセス、cron タスク。

- マイクロサービス間の認証。
- アプリケーションがユーザーに代わってではなく、独自のアカウントとして機能するシナリオ。

OAuth2 | リソースオーナーパスワード認証グラント (RFC 6749)

概要

リソースオーナーパスワード資格情報 (ROPC) グラントにより、クライアントはユーザーのユーザー名とパスワードを直接収集し、アクセストークンと交換できます。ユーザーの資格情報がトークンエンドポイントに送信され、有効な場合はサーバーがアクセストークンを返します。

このグラントタイプは、リソースオーナーとクライアントの間に高い信頼関係がある場合にのみ使用すべきです。例えば、クライアントが認可サーバーを運営する同じ組織によって開発されたファーストパーティアプリケーションである場合などです。

セキュリティ警告

このグラントタイプは他の OAuth2 フローより安全性が低いと見なされています。ユーザーの認証情報をクライアントアプリケーションに直接公開するため、委任された認証の原則に反します。OAuth 2.1 仕様ではこのグラントタイプが非推奨になっています。可能な限り [Authorization Code with PKCE](#) を使用してください。

フロー

1. ユーザーはユーザー名とパスワードをクライアントアプリケーションに直接提供します。
2. クライアントは username、password、client_id、client_secret、grant_type=password、および scope をトークンエンドポイントに POST します。
3. 認可サーバーが認証情報を検証します。
4. サーバーは access_token、token_type、expires_in、およびオプションで refresh_token を返します。

設定

プロパティ	説明
<code>OAuth2Options.GrantType</code>	<code>auth2ResourceOwnerPassword</code> に設定します。
<code>OAuth2Options.ClientId</code>	認可サーバーが発行したクライアント識別子です。
<code>OAuth2Options.ClientSecret</code>	認証サーバーによって発行されたクライアントシークレット。
<code>OAuth2Options.UserName</code>	リソースオーナーのユーザー名。
<code>OAuth2Options.Password</code>	リソースオーナーのパスワード。
<code>AuthorizationServerOptions.TokenURL</code>	トークンエンドポイント URL。ユーザーの認証情報が直接提供されるため、AuthURL は不要です。
<code>AuthorizationServerOptions.Scope</code>	アクセスリクエストのスコープ。

使用例

```
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();
```

トークンのリフレッシュ

サーバーがリフレッシュトークンを返した場合、ユーザーの資格情報を再入力せずに新しいアクセストークンを取得するために使用できます:

使用するタイミング

Resource Owner Password Credentials グラントを使用する用途のみ:

- より安全なフローに移行できないレガシーアプリケーション。
- クライアントと認可サーバーの両方を制御するファーストパーティアプリケーション。
- 直接のユーザー名/パスワード認証から OAuth2 への移行シナリオ。

他のすべてのケースでは、代わりに [PKCE を使用した認可コード](#) または [認可コード](#) を使用してください。

OAuth2 | デバイス認可グラント (RFC 8628)

概要

Device Authorization グラント (Device Code フローとも呼ばれる) は、ブラウザーを簡単に表示できないか、キーボード入力を受け付けられない入力制約のあるデバイス向けに設計されています。スマートテレビ、IoT デバイス、メディアプレーヤー、CLI ツール、キオスクなどが例として挙げられます。デバイスは短いコードを表示し、ユーザーは別のデバイス (電話やコンピューターなど) にそのコードを入力して認証を完了します。

フロー

1. クライアントは `client_id` および `scope` を **DeviceAuthorizationURL** に POST します。
2. サーバーは `device_code`、`user_code`、`verification_uri`、`expires_in`、`interval` を返します。
3. クライアントはユーザーコードと検証 URI とともに **OnDeviceCode** イベントを発生させます。アプリケーションはこれをユーザーに表示します (例: "https://provider.com/device にアクセスしてコード ABCD-1234 を入力してください")。
4. ユーザーは別のデバイスで確認 URI に移動し、ユーザーコードを入力します。
5. ユーザーはセカンダリデバイスで認証してパーミッションを付与します。
6. その間、クライアントは `device_code` を使用して `interval` 秒ごとにトークンエンドポイントを自動的にポーリングします。
7. ユーザーが認可を完了すると、次のポーリングで `access_token` が返されます。
8. ユーザーが承認する前にデバイスコードが期限切れになった場合、**OnDeviceCodeExpired** イベントが発生します。

設定

プロパティ	説明
<code>OAuth2Options.GrantType</code>	auth2DeviceCode に設定します。
<code>OAuth2Options.ClientId</code>	認可サーバーが発行したクライアント識別子です。
<code>OAuth2Options.ClientSecret</code>	オプション。プロバイダーによっては必須の場合と不要な場合があります。
<code>AuthorizationServerOptions.TokenURL</code>	ポーリングに使用されるトークンエンドポイント URL。
<code>AuthorizationServerOptions.DeviceAuthorizationURL</code>	クライアントがデバイスコードをリクエストするデバイス認可エンドポイント。
<code>AuthorizationServerOptions.Scope</code>	アクセスリクエストの範囲。

イベント

イベント	説明
<code>OnDeviceCode</code>	デバイスコードとユーザーコードがサーバーから受信されたときに発生します。ユーザーコードと検証URIをユーザーに表示してください。

<code>OnDeviceCodeExpired</code>	デバイスコードがユーザーが認証を完了する前に期限切れになったときに発生します。アプリケーションはユーザーにフローを再開するよう促す必要があります。
<code>OnAfterAccessToken</code>	ユーザーが認可を完了し、アクセストークンが受信されたときに発生します。
<code>OnErrorAccessToken</code>	デバイスフロー中に回復不可能なエラーが発生したときに発生します。

使用例

デバイスコードイベントの処理

```
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();
```

ポーリング動作

`start` を呼び出した後、コンポーネントはサーバーが指定したインターバル (通常 5 秒) でトークンエンドポイントを自動的にポーリングします。サーバーが `slow_down` エラーで応答した場合、コンポーネントはポーリング間隔を 5 秒増加させます。次のいずれかが発生するまでポーリングが続きます:

- ユーザーが認可を完了し、トークンが返されます。
- デバイスコードが期限切れになります (`OnDeviceCodeExpired` が発生します)。
- 回復不可能なエラーが発生した場合 (`OnErrorAccessToken` が発生)。

使用するタイミング

以下の場合に Device Authorization グラントを使用します:

- スマート TV とストリーミング デバイス。
- コマンドラインインターフェイス (CLI) ツール。
- 入力機能が限られた IoT デバイス。
- キオスクとデジタルサイネージ。
- 完全な URL や資格情報を入力するのが難しいデバイス。

OAuth2 | DPoP - Demonstrating Proof of Possession (RFC 9449)

概要

DPoP (Demonstrating Proof of Possession) は、秘密鍵の所有証明を要求することでアクセストークンを特定のクライアントにバインドするセキュリティメカニズムです。誰でも使用できるベアラートークンとは異なり、DPoP バインドトークンは対応する秘密鍵なしでは使用できません。これにより、トークンの盗難やリプレイ攻撃に対する保護が提供されます。

DPoP はそれ自体グラントタイプではなく、任意の OAuth2 グラントタイプ (Authorization Code、Client Credentials など) で使用できるセキュリティ強化機能です。

仕組み

1. クライアントが非対称キーペア (ES256 または RS256) を生成します。
2. クライアントは以下の構造で **DPoP プルーフ JWT** を作成します:
 - ヘッダー: `typ: dpop+jwt`、`alg: ES256`、`jwk: {public key}`
 - ペイロード: `htm` (HTTP メソッド)、`htu` (HTTP URI)、`iat` (発行日時)、`jti` (一意識別子)
3. クライアントは DPoP プルーフを HTTP ヘッダー (`DPoP: <proof>`) としてトークンリクエストとともに送信します。
4. 認可サーバーは発行されたトークンをクライアントの公開鍵の JWK サムプリントにバインドします。
5. サーバーは `token_type: Bearer` の代わりに `token_type: DPoP` を返します。
6. 後続の API 呼び出しでは、クライアントは `Authorization: DPoP <token>` ヘッダーと、そのリクエスト固有の新しい `DPoP: <proof>` ヘッダーの両方を送信します。

キー生成

GenerateDPoPKeyPair メソッドを使用して、Delphi で直接 ES256 キーペアを生成できます。このメソッドは内部で OpenSSL を使用してキーペアを作成し、`PrivateKey` と `PublicKeyJWK` の両方を自動的に設定します:

```
OAuth2->DPoPOptions->Enabled = true;  
OAuth2->DPoPOptions->Algorithm = dpopES256;  
OAuth2->GenerateDPoPKeyPair(); // Generates PrivateKey + PublicKeyJWK automatically  
OAuth2->Start();
```

PublicKeyToJWK — 任意のキーまたは証明書を JWK に変換

すでに公開鍵、秘密鍵、または X.509 証明書 (PEM 形式) をお持ちの場合は、**PublicKeyToJWK** クラス関数を使用して JWK に変換できます。これらの入力のいずれかを受け付け、正しい JWK JSON を返します:

入力	PEM Header
EC または RSA 公開鍵	-----BEGIN PUBLIC KEY-----
EC 秘密鍵	-----BEGIN EC PRIVATE KEY-----
RSA 秘密鍵	-----BEGIN RSA PRIVATE KEY-----
X.509 証明書	-----BEGIN CERTIFICATE-----

PEMキーファイルから

```
// 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"));
```

スマートカード証明書から

スマートカードとHSMは秘密鍵を保護しており、読み取りやエクスポートはできません。ただし、**証明書**（公開鍵を含む）は常にエクスポート可能です。証明書をPEMとしてエクスポートし、`PublicKeyToJWK` に渡してください:

```
// 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;
```

このアプローチは、証明書をエクスポートできる任意のカードや HSM で動作します: PKCS#11 トークン、Windows CNG スマートカード、USB セキュリティキー (YubiKey、Feitian)、クラウド HSM (Azure Key Vault、AWS CloudHSM)。

OpenSSLコマンドラインツールの使用

外部でキーペアを生成することもできます:

DPoP オプション

プロパティ	説明
<code>DPoPOptions.Enabled</code>	DPoP 証明の生成を有効にするには True に設定します。
<code>DPoPOptions.Algorithm</code>	署名アルゴリズム。サポートされる値: ES256 (推奨)、 RS256 。
<code>DPoPOptions.PrivateKey</code>	DPoP 証明 JWT に署名するために使用される PEM エンコードされた秘密鍵。
<code>DPoPOptions.PublicKeyJWK</code>	DPoP ブルーフヘッダーに含まれる JWK (JSON Web Key) フォーマットの公開鍵。

メソッド

メソッド	説明
<code>GenerateDPoPKeyPair</code>	OpenSSL を使用して ES256 キーペアを生成し、 <code>DPoPOptions.PrivateKey</code> (PEM) と <code>DPoPOptions.PublicKeyJWK</code> (JSON) の両方を自動的に設定します。

<code>GetDPoPProof(Method, URL, AccessToken)</code>	指定された HTTP メソッド、URL、およびアクセストークンに対する DPoP プルーフ JWT を生成します。署名済み JWT 文字列を返します。プルーフにはトークンをプルーフにバインドするための <code>ath</code> クレーム（アクセストークンハッシュ）が含まれます。
<code>GetDPoPJWKThumbprint</code>	公開鍵の RFC 7638 JWK サンプリント（SHA-256）を返します。サーバーがトークンをバインドするために使用します。

ノンス処理

一部の認証サーバーはDPoP証明書にサーバーが提供するノンスを必要とします。サーバーがDPoP-Nonceヘッダーと400または401ステータスで返答すると、コンポーネントは自動的に次を実行します：

1. DPoP-Nonce レスポンスヘッダーからノンスを抽出します。
2. 次のDPoP証明JWTの `nonce` クレームにnonceを含めます。
3. 更新された証明でリクエストを再試行します。

OnDPoPSign を使用したカスタム署名（HSM / スマートカード）

一部の環境では、秘密鍵がハードウェアセキュリティモジュール（HSM）またはスマートカード（例: PKCS#11 やWindows CNG経由）に保存されており、**抽出できません**。OnDPoPSignイベントを使用すると、コンポーネントが他のすべて（プルーフ構造、nonce、ヘッダー）を処理する間、署名を外部で実行できます。

OnDPoPSign が割り当てられている場合、コンポーネントは内部署名を試みる前にそれを呼び出します。Handled := True を設定して Signature を提供すると、コンポーネントはあなたの署名を使用し、内部署名をスキップします。この場合、DPoPOptions.PrivateKey プロパティは不要です。

イベントシグネチャ

パラメータ	方向	説明
<code>SigningInput</code>	in	署名対象データ: <code>Base64URL(header) + "." + Base64URL(payload)</code> 。これは標準的な JWS 署名入力です。
<code>Algorithm</code>	in	アルゴリズム名: <code>"ES256"</code> または <code>"RS256"</code> 。
<code>Signature</code>	out	署名デバイスが生成した Base64URL エンコードされた署名バイトに設定します。
<code>Handled</code>	out	カスタム署名を使用するには <code>True</code> に設定します。 <code>False</code> の場合、コンポーネントは <code>PrivateKey</code> を使用した内部署名にフォールバックします。

スマートカードの例

次の例は、署名カードでDPoPを使用する方法を示しています。秘密鍵はカードから決して離れません。ハッシュのみが署名のためにカードに送信されます。

```
// 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;
}
```

このアプローチは、PKCS#11トークン、Windows CNG（CryptoAPI Next Generation）、Azure Key Vault、AWS CloudHSM、またはECDSA/RSA署名を生成できる任意のカスタムハードウェアなど、あらゆる署名バックエンドで機能します。

使用例

```
// 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

このコンポーネントはサーバーサイドコンポーネントに OAuth2 プロトコルの実装を提供します。

はじめに

このコンポーネントはサーバーサイドコンポーネントに OAuth2 プロトコルの実装を提供します。サーバーコンポーネントには、Authorization.OAuth.OAuth2 というプロパティがあり、ここに TsgcHTTP_OAuth2_Server のインスタンスを割り当てることができます。したがって、Authentication が有効で OAuth2 プロパティが OAuth2 Server コンポーネントにアタッチされている場合、WebSocket および HTTP リクエストは処理されるために Bearer Token を必要とし、ない場合は接続が自動的に閉じられます。

```
TsgcHTTP_OAuth2_Server *OAuth2 = new TsgcHTTP_OAuth2_Server(this);
Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2 = OAuth2;
```

サーバーは以下の認証タイプをサポートしています。

- **auth2Code:** シングルページアプリケーション、ウェブアプリケーション、ネイティブインストールアプリケーションを含む大多数のアプリケーションタイプで認証と認可を行うために使用されます。このフローにより、アプリはリソースへのアクセスに使用できる access_token、追加の access_token を取得するためのリフレッシュトークン、およびサインインユーザーの ID トークンを安全に取得できます。
- **auth2ClientCredentials:** このグラントタイプは、ユーザーとの直接的なインタラクションなしにバックグラウンドで実行されるサーバー間インタラクションによく使用されます。この種のアプリケーションはデーモンまたはサービスアカウントと呼ばれることがよくあります。
- **password** (リソースオーナーパスワード認証情報): ユーザーの認証情報を直接処理してサインインするアプリケーションを許可します。クライアントはユーザーのユーザー名とパスワードをトークンエンドポイントに送信します。
- **urn:ietf:params:oauth:grant-type:device_code** (デバイスコード): RFC 8628 に基づくデバイス認可グラント。入力が制限されたデバイス (スマートテレビ、IoT デバイス) がセカンダリデバイスでユーザーに認可させることでユーザー認可を取得できるようにします。

アプリの登録時に認可タイプをカスタマイズできます。デフォルトでは、すべての認可タイプがサポートされます。

エンドポイント

デフォルトでは、コンポーネントは以下のエンドポイントで認可とトークンのリクエストを処理するように設定されています

Authorization: /sgc/oauth2/auth

Token: /sgc/oauth2/token

Revocation : /sgc/oauth2/revoke

Introspection: /sgc/oauth2/introspect

デバイス認証: /sgc/oauth2/device

デバイス検証: /sgc/oauth2/device/verify

したがって、サーバーがポート 443 でリッスンし、ドメインが www.esegece.com の場合、EndPoints は次のようになります:

Authorization: <https://www.esegece.com/sgc/oauth2/auth>

トークン: <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>

デバイス確認: <https://www.esegece.com/sgc/oauth2/device/verify>

エンドポイントは OAuth2Options プロパティで設定できます。

デフォルトでは、PKCE (CSRF および認証コードインジェクション攻撃を防ぐための Authorization Code フローの拡張機能) が有効になっています。

設定

OAuth2プロセスを開始する前に、どのアプリが利用可能になるかを登録する必要があります。これはOAuth2サーバーコンポーネントのAppsプロパティを使用していきます。

Register App

新しいアプリケーションを OAuth2 サーバーに追加するには Apps.AddApp を使用します。次のパラメータを設定する必要があります：

- **アプリ名:** アプリケーションの名前です。例: MyApp
- **RedirectURI:** はレスポンスがリダイレクトされる場所です。例: http://127.0.0.1:8080
- **ClientId:** 公開情報であり、クライアントの ID です。
- **ClientSecret:** 機密に保つ必要があります。

オプションで以下のパラメータを設定できます。

- **ExpiresIn:** デフォルトは 3600 秒なので、トークンは 1 時間後に期限切れになります。必要に応じて大きな値を設定できます。
- **RefreshToken:** デフォルトでは更新トークンがサポートされています。サポートしない場合は、このパラメータを false に設定してください。
- **AllowedGrantTypes:** デフォルトでは、すべてのグラントタイプがサポートされています (auth2Code と auth2ClientCredentials)。ただし、サーバーはコード認可のみ、またはクライアント認証情報のみを許可するように設定できます。

アプリを削除

Apps.RemoveApp を使用して既存のアプリを削除します。

AddToken

トークンが発行されている間にサーバーが再起動された場合、OAuth2 Serverを起動する前、かつAppを登録した後に、AddTokenメソッドを使用してこれらのトークンを復元できます

- **AppName:** アプリケーションの名前。
- **Token:** アクセストークン。
- **Expires:** トークンの有効期限。
- **RefreshToken:** リフレッシュ トークン。

RemoveToken

発行済みトークンを削除します。

OAuth2Options

OAuth2Options プロパティを使用して、サーバーエンドポイントとオプション機能を設定できます。

失効

[RFC 7009](#)によるトークン取り消し。有効にすると、クライアントは以前に発行されたアクセストークンまたはリフレッシュトークンを取り消すことができます。

- **OAuth2Options.Revocation.Enabled:** True に設定すると、失効エンドポイントが有効になります。
- **OAuth2Options.Revocation.URL:** エンドポイントの URL パス。デフォルト: /sgc/oauth2/revoke

イントロスペクション

[RFC 7662](#) によるトークンイントロスペクション。有効にすると、リソースサーバーは認可サーバーにクエリしてトークンのアクティブ状態とメタデータを確認できます。

- **OAuth2Options.Introspection.Enabled:** イントロスペクションエンドポイントを有効にするには True に設定します。
- **OAuth2Options.Introspection.URL:** エンドポイントの URL パス。デフォルト: /sgc/oauth2/introspect

DeviceAuthorization

[RFC 8628](#) によるデバイス認可グラント。有効にすると、入力制限デバイスはセカンダリデバイスでユーザーが認可することで認可をリクエストできます。

- **OAuth2Options.DeviceAuthorization.Enabled:** デバイス認可エンドポイントを有効にするにはTrueに設定します。
- **OAuth2Options.DeviceAuthorization.URL:** デバイスコードリクエストのエンドポイント URL パス。デフォルト: /sgc/oauth2/device
- **OAuth2Options.DeviceAuthorization.VerificationURL:** ユーザー確認ページのエンドポイント URL パス。デフォルト: /sgc/oauth2/device/verify
- **OAuth2Options.DeviceAuthorization.ExpiresIn:** デバイスコードの有効期間(秒)。デフォルト: 600(10分)。
- **OAuth2Options.DeviceAuthorization.Interval:** クライアントがトークンエンドポイントをポーリングする際に使用すべき最小ポーリング間隔 (秒単位)。デフォルト値: 5。

リファレンス

- プロパティ — コンポーネントに 2 つのプロパティがあります。
- メソッド — 11個のパブリックメソッドがあります。
- イベント — 処理できる13個のイベント。

ガイド

- クイックスタート
 - [OAuth2 サーバーの例](#)
 - [OAuth2 サインイン HTML のカスタマイズ](#)
 - [OAuth2 サーバーエンドポイント](#)
 - [OAuth2 アプリの登録](#)
 - [OAuth2 アクセストークンの回復](#)
- 認証
 - [OAuth2 サーバー認証](#)
 - [一部の URL を OAuth2 認証から除外する](#)

OAuth2 | サーバー 使用例

[TsgcWebSocketHTTPServer](#) を使用したシンプルな OAuth2 サーバーの例を作成してみましょう。

まず、ポート 443 で リッスン し、sgc.pem ファイル の 自己署名証明書 を使用する新しい TsgcWebSocketHTTPServer を作成します。

```
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;
```

次に TsgcHTTP_OAuth2_Server の新しいインスタンスを作成し、事前に作成したサーバーに割り当てます。次の値で新しいアプリケーションを登録します。

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;
```

次に OAuth2 サーバーコンポーネントの OnOAuth2Authentication イベントを処理し、ユーザーをログインさせる独自のメソッドを実装します。ここでは "user/secret" のペアを使用してログインを受け入れます。

```
void OnOAuth2Authentication(TsgcWSConnection *Connection, TsgcHTTPOAuth2Request *OAuth2, string aUser,
    string aPassword, ref bool Authenticated)
{
    if ((aUser == "user") and (aPassword == "secret"))
    {
        Authenticated = true;
    }
}
```

最後に、サーバーを起動して OAuth2 クライアントを使用してログインします。例えば、sgcWebSockets ライブラリに含まれる [TsgcHTTP_OAuth2_Client](#) を使用できます。

OAuth2

Configuration

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

Redirect URL

Port

8080

Local Server Options

OAuth2 Options

ClientId

client-id

Secret

client-secret

Username

Password

New Access Token

Access Token

Token Type

Expires In

Refresh Token

Scope

Refresh Token

新しいアクセストークンをリクエストします。新しい Web ブラウザーセッションが表示され、ユーザーは接続を許可してからログインする必要があります。

OAuth2 Authorization

Sign in

Introduce your username and password.

GO BACK

Sign in

SIGN IN

ログインが成功すると、新しいトークンがクライアントに返されます。その後のすべてのリクエストには、HTTP ヘッダーにこの Bearer トークンを含める必要があります。

 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

OAuth2 Options

ClientId

client-id

Secret

client-secret

Username

Password

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 | サインイン HTML のカスタマイズ

OAuth2クライアントが新しいAccess Tokenを取得するリクエストを行うと、この接続を許可し、ユーザーとパスワードでログインするためのWebページがWebブラウザに表示されます。HTML ページ は デフォルト で サーバー コンポーネント に 含まれて いますが、このコードは **OnAuth2BeforeDispatchPage** イベントを使用してカスタマイズできます。

```
void OnAuth2BeforeDispatchPage(TObject *Sender; TsgcHTTPOAuth2Request *OAuth2; ref string HTML)
{
    HTML = "your custom html";
}
```

完全に新しい HTML コードで HTML をカスタマイズする場合は、少なくともユーザー名とパスワードが送信されるフォームを維持する必要があります:

```
<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>サインイン</button>
</form>
```

非表示の id パラメーターは、サーバーがリクエストを識別できるように 元のフォームと同じ値を保持する必要があります。

OAuth2 | サーバー エンドポイント

デフォルトでは、OAuth2 サーバーは以下のエンドポイントを使用します:

Authorization: /sgc/oauth2/auth

Token: /sgc/oauth2/token

Revocation: /sgc/oauth2/revoke (POST) - [RFC 7009](#) に従ってトークンを失効させます。

Introspection: /sgc/oauth2/introspect (POST) - [RFC 7662](#) に基づくトークンメタデータを返します。

デバイス認証: /sgc/oauth2/device (POST) - [RFC 8628](#) に従ってデバイスコードを発行します

Device Verification: /sgc/oauth2/device/verify (GET/POST) - ユーザー検証ページ

つまり、サーバーが IP 80.54.41.30 のポート 8443 でリッスンしている場合、完全な OAuth2 エンドポイントは次のようになります:

Authorization: https://80.54.41.30:8443/sgc/oauth2/auth

Token: https://80.54.41.30:8443/sgc/oauth2/token

失効: 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

デバイス検証: https://80.54.41.30:8443/sgc/oauth2/device/verify

これらのエンドポイントは簡単に変更できます。コンポーネントの `OAuth2Options` プロパティにアクセスし、Authorization URL と Token URL を変更するだけです。

例: エンドポイントが必要な場合は

Authorization: https://80.54.41.30:8443/authentication/auth

Token: https://80.54.41.30:8443/authentication/token

`OAuth2Options` プロパティを以下の値で設定します:

`OAuth2Options.Authorization.URL` = /authentication/auth

`OAuth2Options.Token.URL` = /authentication/token

同じアプローチが他のエンドポイントにも適用されます:

`OAuth2Options.Revocation.URL` = /authentication/revoke

`OAuth2Options.Introspection.URL` = /authentication/introspect

`OAuth2Options.DeviceAuthorization.URL` = /authentication/device

`OAuth2Options.DeviceAuthorization.VerificationURL` = /authentication/device/verify

OAuth2 | アプリの登録

クライアントによって新しい OAuth2 がリクエストされる前に、アプリをサーバーに登録する必要があります。新しいAppを登録するには、次の情報が必要です:

- **アプリ名:** アプリケーションの名前です。例: MyApp
- **RedirectURI:** はレスポンスがリダイレクトされる場所です。例: http://127.0.0.1:8080
- **ClientId:** 公開情報であり、クライアントの ID です。
- **ClientSecret:** 機密に保つ必要があります。

オプションで以下のパラメーターを設定できます。

- **ExpiresIn:** デフォルトは 3600 秒なので、トークンは 1 時間後に期限切れになります。必要に応じて大きな値を設定できます。
- **RefreshToken:** デフォルトでは更新トークンがサポートされています。サポートしない場合は、このパラメーターを `false` に設定してください。

新しいクライアントが OAuth2 を使用して認証する場合、まずアプリをサーバーに登録する必要があります。以下を使用できます:

1. RegisterApp

このメソッドにはアプリ名とRedirectURIが必要であり、ClientIdとClientSecretを返します。

2. Apps.AddApp

このメソッドにはAppName、RedirectURI、ClientId、ClientSecret が必要です。通常、サーバーにすでに作成されたアプリがある場合に、サーバーが起動する前にそれらをロードするためにこのメソッドを使用できます。

どちらのメソッドも同じことをします。サーバーにアプリケーションを登録しますが、最初のメソッドはアプリが初めて登録される場合に最も有用で、2 番目のメソッドはサーバーを起動する前にすべての登録済みアプリをロードしたい場合（例えばデータベースに保存されている場合）に使用します。

OAuth2 | アクセストークンの回復

OAuth2 サーバーが破棄（再起動のため）され、有効なアクセストークンが発行されている場合、デフォルトではこれらは失われます。**AddToken** メソッドを使用してこれらのアクセストークンを復元できます。このメソッドはトークンを OAuth2 サーバーに保存します。

トークンを追加するには以下の情報が必要です。

- **AppName** : アプリ名。
- **Token** : アクセストークン。
- **Expires**: トークンの有効期限。
- **RefreshToken**: リフレッシュ トークン。

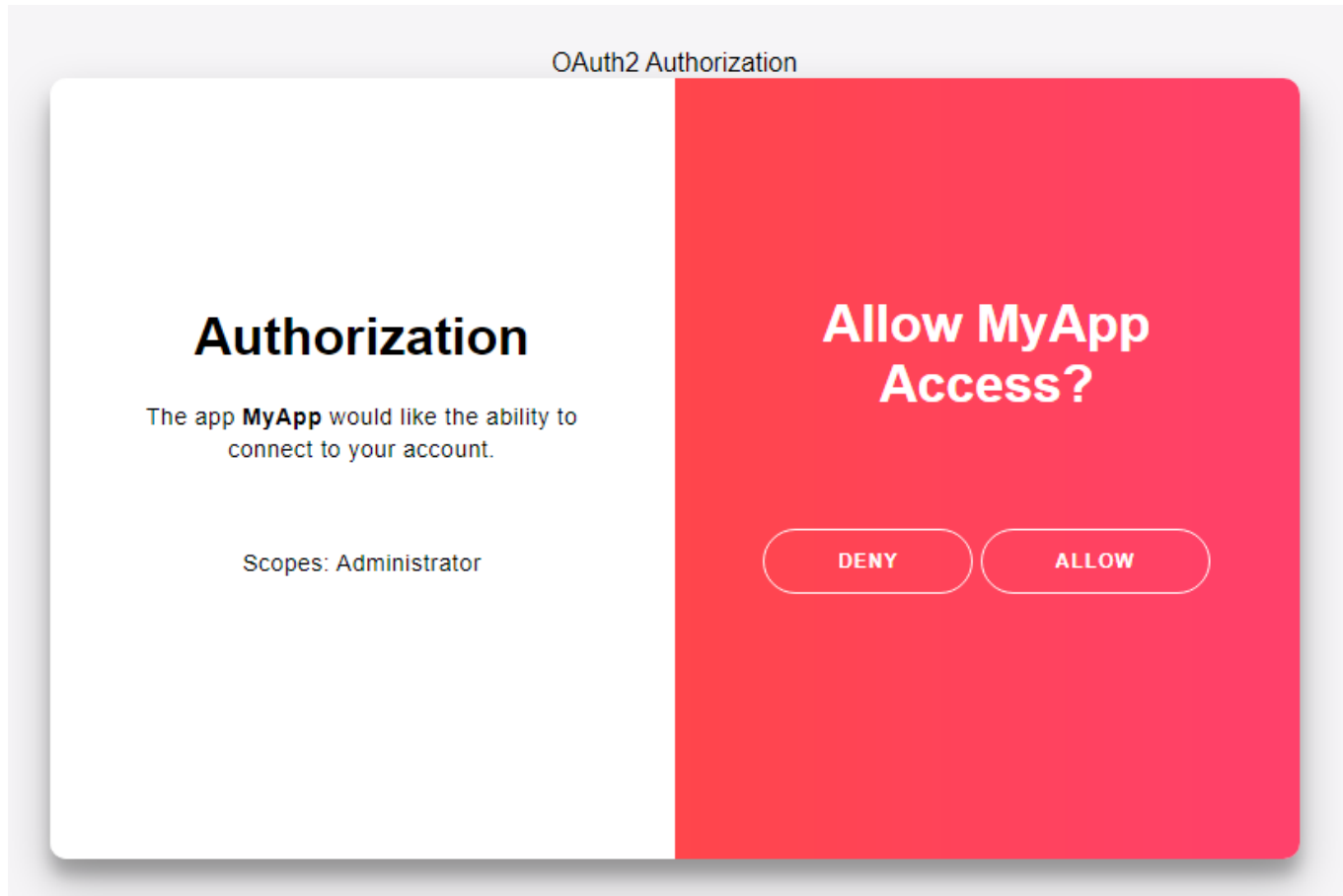
OAuth2AfterAccessToken イベントを処理することで、発行されたトークンを保存できます。

```
private void OnOAuth2AfterAccessToken(TObject *Sender, TsgcWSConnection *Connection, TsgcHTTPOAuth2Request *OAuth2Request, TsgcHTTPOAuth2Response *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 | サーバー認証

OAuth2 クライアントが新しい認証をリクエストすると、サーバーはユーザーが接続を許可してログインする必要がある Web ページを表示します。このページは `sgcWebSockets` ライブラリによって提供され、クライアントが認証をリクエストしたときに自動的にディスパッチされます。



ユーザーがアクセスを許可すると、ユーザー名とパスワードを入力するログインフォームが表示されます。このデータは `OnOAuth2Authentication` イベントで受信されるため、ユーザー/パスワードが正しいかどうかを検証し、正しければ `Authenticated` パラメーターを `true` に設定する必要があります。

```
void OnOAuth2Authentication(TsgcWSConnection *Connection, TsgcHTTPOAuth2Request *OAuth2, string aUser,
    string aPassword, ref bool Authenticated)
{
    if ((aUser == "user") and (aPassword == "secret"))
    {
        Authenticated = true;
    }
}
```

OAuth2 | 認証なし URL

デフォルトでは、サーバーサイドで OAuth2 が有効になっている場合、すべての HTTP リクエストにベアラー トークンによる認証が必要です。

Bearer Token を使用せずにアクセスできる URL を許可したい場合は、**OnOAuth2BeforeRequest** イベントを使用できます。

```
procedure OnOAuth2BeforeRequest(TObject *Sender; TsgwSConnection *aConnection; TStringList *aHeaders;  
    ref bool Cancel)  
{  
    if (DecodeGETFullPath(aHeaders) == "/Public.html")  
    {  
        Cancel = true;  
    }  
}
```


OAuth2 サーバー | 認可コードグラント

概要

Authorization Code グラントは、サーバーサイドアプリケーションで最も一般的な OAuth2 フローです。[TsgcHTTP_OAuth2_Server](#)は、このフローを処理するための2つのエンドポイントを提供します: ユーザーにサインインページを表示する Authorization エンドポイントと、認可コードをアクセストークンと交換する Token エンドポイントです。

このフローは、クライアントがブラウザとやり取りできる Web アプリケーション、デスクトップアプリケーション、およびモバイルアプリケーションに適しています。

フロー

1. クライアントはユーザーを `/sgc/oauth2/auth?`
`response_type=code&client_id=...&redirect_uri=...&scope=...&state=...` にリダイレクトします
2. サーバーは HTML サインインページのカスタマイズを許可するために **OnOAuth2BeforeDispatchPage** を発生させます。
3. ユーザーが資格情報を送信します。サーバーは **OnOAuth2Authentication** を発生させてユーザー名とパスワードを検証します。
4. 認証が成功した場合、サーバーは `?code=...&state=...` を付加して `redirect_uri` にユーザーをリダイレクトします。
5. クライアントは `/sgc/oauth2/token` に
`grant_type=authorization_code&code=...&redirect_uri=...&client_id=...&client_secret=...` で POST します。
6. サーバーは認可コードとクライアント資格情報を検証し、**OnOAuth2AfterAccessToken** を発火させ、`access_token`、`token_type`、`expires_in`、およびオプションで `refresh_token` を含む JSON レスポンスを返します。

設定

プロパティ	説明
<code>OAuth2Options.Authorization.Enabled</code>	認可エンドポイントを有効にするには True に設定します。デフォルト: True。
<code>OAuth2Options.Authorization.URL</code>	認可エンドポイントの URL パス。デフォルト: <code>/sgc/oauth2/auth</code>
<code>OAuth2Options.Token.Enabled</code>	True に設定するとトークンエンドポイントを有効にします。デフォルト値: True。
<code>OAuth2Options.Token.URL</code>	トークンエンドポイントの URL パス。デフォルト: <code>/sgc/oauth2/token</code>
<code>OAuth2Options.PKCE</code>	True の場合、サーバーは認証コードリクエストに PKCE (<code>code_challenge</code> と <code>code_verifier</code>) を要求します。デフォルト: True。

アプリ登録

クライアントが Authorization Code フローを使用できるようにする前に、サーバーにアプリケーションを登録する必要があります。 `Apps.AddApp` を使用し、`AllowedGrantTypes` に `auth2Code` が含まれていることを確認します。

```
OAuth2Server->Apps->AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    TsgcOAuth2GrantTypes() << auth2Code);
```

イベント

イベント	説明
<code>OnOAuth2BeforeDispatchPage</code>	サインインHTMLページがユーザーへ送信される前に発生します。HTMLコンテンツのカスタマイズを可能にします。
<code>OnOAuth2Authentication</code>	ユーザーが資格情報を送信したときに発生します。ここでユーザー名とパスワードを検証し、 <code>Authenticated</code> を <code>True</code> または <code>False</code> に設定してください。
<code>OnOAuth2AfterAccessToken</code>	サーバーがアクセストークンを正常に発行した後に発生します。ログ記録に役立ちます。
<code>OnOAuth2AfterValidateAccessToken</code>	クライアントが有効期限内のトークンでリクエストを行った際に発生します。ログ記録および監査に役立ちます。

使用例

次の例は、アプリ登録とイベントハンドラーを含む、Authorization Codeグラントを使用した完全なサーバーセットアップを示しています。

```
// 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->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 サポート

`OAuth2Options.PKCE` が `True` (デフォルト) に設定されている場合、サーバーは PKCE を適用します。クライアントは認可リクエストに `code_challenge` パラメーターを含め、トークンリクエストに対応する `code_verifier` を含める必要があります。サーバーはトークンを発行する前に `code_verifier` が元の `code_challenge` と一致することを確認します。

PKCE は特にクライアントシークレットを安全に保存できないパブリッククライアント (ネイティブアプリや SPA) を含む、すべてのクライアントに推奨されます。

OAuth2 サーバー | クライアント資格情報グラント

概要

クライアント資格情報グラントは、ユーザーインタラクションが不要なマシン間（M2M）通信に使用されます。クライアントは独自の資格情報（`client_id` と `client_secret`）を使用して [TsgcHTTP_OAuth2_Server](#) に直接認証し、アクセストークンを受け取ります。認可コードステップ、リダイレクト、ユーザーログインページはありません。

このグラントタイプは、クライアントアプリケーションが特定のユーザーの代わりにではなく、自身の代わりに動作する場合に適しています。

フロー

- クライアントは `grant_type=client_credentials&client_id=...&client_secret=...&scope=...` でトークンエンドポイントに POST します
- サーバーは登録済みアプリケーションに対してクライアントの認証情報を検証します。
- サーバーは **OnOAuth2AfterAccessToken** を発生させ、`access_token`、`token_type`、`expires_in` を含む JSON レスポンスを返します。

このグラントタイプでは認証するユーザーがいないため、`OnOAuth2Authentication` イベントは発生しません。サーバーは登録済みアプリに対して `client_id` と `client_secret` を自動的に検証します。

設定

`auth2ClientCredentials` を含む `AllowedGrantTypes` でアプリケーションを登録します。このフローにはトークンエンドポイントのみが必要です。

プロパティ	説明
<code>OAuth2Options.Token.Enabled</code>	True に設定するとトークンエンドポイントを有効にします。デフォルト値: True。
<code>OAuth2Options.Token.URL</code>	トークンエンドポイントの URL パス。デフォルト: <code>/sgc/oauth2/token</code>

アプリ登録

```
OAuth2Server->Apps->AddApp("MyServiceApp", "",
    "service-client-id", "service-client-secret", 3600, false,
    TsgcOAuth2GrantTypes() << auth2ClientCredentials);
```

`RedirectURI` はリダイレクトが不要なため空のままにできます。クライアントはいつでも資格情報を使用して新しいトークンをリクエストできるため、`RefreshToken` は通常 `False` に設定されます。

イベント

イベント	説明
<code>OnOAuth2AfterAccessToken</code>	サーバーがアクセストークンを正常に発行した後に発生します。ログ記録に役立ちます。

OnOAuth2AfterValidateAccessToken

クライアントが有効なトークンでリクエストを行ったときに発生します。監査に役立ちます。

使用例

```
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);
}
```

使用するタイミング

サーバーのクライアント認証情報グラントを次の用途に使用します。

- ユーザーインタラクションなしに認証するバックエンドサービスとデーモン。
- サーバー間 API 通信。
- マイクロサービス間の認証。
- 自動化されたプロセス、スケジュールされたジョブ、バッチ操作。

OAuth2 サーバー | リソースオーナーパスワード資格情報グラント

概要

リソースオーナーパスワードクレデンシャル (ROPC) グラントは、クライアントがユーザーのユーザー名とパスワードをトークンエンドポイントに直接送信できるようにします。[TsgcHTTP_OAuth2_Server](#) は **OnOAuth2Authentication** イベントを通じてクレデンシャルを検証し、認証が成功した場合はアクセストークンを発行します。

このフローにはブラウザーのリダイレクトやサインインページは含まれません。クライアントはユーザーから直接資格情報を収集してサーバーに送信します。

セキュリティ警告

このグラントタイプは **OAuth 2.1 では非推奨です**。Authorization Code with PKCE などの他のフローが実行可能な場合（例: レガシーアプリケーションや高度に信頼された第一者クライアント）にのみ使用してください。PKCE を使用した Authorization Code グラントが推奨される代替手段です。

フロー

- クライアントは `grant_type=password&username=...&password=...&client_id=...&client_secret=...&scope=...` でトークンエンドポイントに POST します
- サーバーは、ユーザー名とパスワードを検証するために **OnOAuth2Authentication** を発生させます。
- 認証された場合、サーバーは **OnOAuth2AfterAccessToken** を発生させ、`access_token`、`token_type`、`expires_in`、オプションで `refresh_token` を含む JSON レスポンスを返します。
- 認証が失敗した場合、サーバーはエラーレスポンスを返します。

設定

`auth2ResourceOwnerPassword` を含む `AllowedGrantTypes` でアプリケーションを登録します。

プロパティ	説明
<code>OAuth2Options.Token.Enabled</code>	True に設定するとトークンエンドポイントを有効にします。デフォルト値: True。
<code>OAuth2Options.Token.URL</code>	トークンエンドポイントの URL パス。デフォルト: <code>/sgc/oauth2/token</code>

アプリ登録

```
OAuth2Server->Apps->AddApp("MyApp", "",
    "my-client-id", "my-client-secret", 3600, true,
    TsgcOAuth2GrantTypes() << auth2ResourceOwnerPassword);
```

イベント

イベント	説明

<code>OnOAuth2Authentication</code>	サーバーがユーザー名とパスワードを受信したときに発火します。ここで資格情報を検証し、 <code>Authenticated</code> を True または False に設定します。
<code>OnOAuth2AfterAccessToken</code>	サーバーがアクセストークンを正常に発行した後に発生します。ログ記録に役立ちます。
<code>OnOAuth2AfterValidateAccessToken</code>	有効なトークンでクライアントがリクエストを行ったときに発生します。

使用例

```
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);
}
```

使用するタイミング

Resource Owner Password Credentials グラントを使用する条件：

- ・クライアントは、高度に信頼されたファーストパーティアプリケーションです。
- ・他のグラントタイプ（PKCEを使用した認可コード）は実現できません。
- ・ユーザーのクレデンシャルを直接収集する既存のレガシーアプリケーションの移行。

新しいアプリケーションでは、常に PKCE を使用した [Authorization Code](#) グラントを優先してください。

OAuth2 サーバー | リフレッシュトークングラント

概要

リフレッシュトークングラントにより、クライアントはユーザーの操作を必要とせず、以前に発行されたリフレッシュトークンを新しいアクセストークンと交換できます。[TsgcHTTP_OAuth2_Server](#) はリフレッシュトークンを検証し、新しいアクセストークン（オプションで新しいリフレッシュトークン）を発行します。これはアクセストークンの有効期限が短い、クライアントが保護されたリソースへの継続的なアクセスが必要な長期セッションに便利です。

フロー

1. クライアントは `grant_type=refresh_token&refresh_token=...&client_id=...&client_secret=...&scope=...` を含むトークンエンドポイントに POST します
2. サーバーはリフレッシュトークンを保存されたトークンと照合して検証します。
3. サーバーは **OnOAuth2AfterRefreshToken** を発生させ、新しい `access_token`、`token_type`、`expires_in`、およびオプションで新しい `refresh_token` を含む JSON レスポンスを返します。

設定

リフレッシュトークンはアプリを登録する際にアプリケーションごとに有効化されます。`Apps.AddApp` の `RefreshToken` パラメータを `True` に設定します。

プロパティ	説明
<code>OAuth2Options.Token.Enabled</code>	True に設定するとトークンエンドポイントを有効にします。デフォルト値: True。
<code>OAuth2Options.Token.URL</code>	トークンエンドポイントの URL パス。デフォルト: <code>/sgc/oauth2/token</code>
<code>RefreshToken</code> (<code>AddApp</code> parameter)	このアプリケーションのリフレッシュトークンを発行するには True に設定します。デフォルト: True。

アプリ登録

アプリを登録する際、`RefreshToken` パラメータを `True` に設定して、サーバーがアクセストークンと共にリフレッシュトークンを発行するようにします。

```
// RefreshToken = true (6th parameter)
OAuth2Server->Apps->AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    TsgcOAuth2GrantTypes() << auth2Code);
```

イベント

イベント	説明

<code>OnOAuth2AfterRefreshToken</code>	サーバーがリフレッシュトークンから新しいアクセストークンを正常に発行した後に発生します。トークンローテーションのログ記録に役立ちます。
<code>OnOAuth2AfterAccessToken</code>	新しいアクセストークンが発行された後に発生します。

使用例

```
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);
}
```

トークンローテーション

リフレッシュトークンが使用されると、サーバーは新しいアクセストークンとともに新しいリフレッシュトークンを発行する場合があります。古いリフレッシュトークンは無効化されます。これはリフレッシュトークンのローテーションと呼ばれ、トークンのリプレイ攻撃を防ぐのに役立ちます。

再起動後のトークンの回復

トークンがまだ有効な状態でサーバーが再起動した場合は、サーバーを起動する前に `Apps.AddToken` を使用してトークンを回復できます。詳細は [OAuth2 アクセストークンの回復](#) を参照してください。

OAuth2 サーバー | デバイス認可グラント (RFC 8628)

概要

デバイス認可グラントは、ブラウザを簡単に表示できないか、キーボード入力を受け付けられない入力制約付きデバイス（スマートTV、IoT デバイス、CLI ツール）向けに設計されています。[TsgcHTTP_OAuth2_Server](#) はデバイス認可エンドポイントと検証ページを提供するため、ユーザーは電話やコンピューターなどのセカンダリデバイスからデバイスを認可できます。

フロー

1. デバイスは `/sgc/oauth2/device` に `client_id=...&scope=...` で POST します。
2. サーバーは **OnOAuth2DeviceAuthorization** を発生させ、`device_code`、`user_code`、`verification_uri`、`expires_in`、`interval` を含む JSON レスポンスを返します。
3. デバイスはユーザーに `user_code` と `verification_uri` を表示します（例: 「<https://example.com/sgc/oauth2/device/verify> にアクセスしてコード ABCD-1234 を入力してください」）。
4. ユーザーはセカンダリデバイスで確認 URI にアクセスし、`user_code` を入力します。
5. サーバーは **OnOAuth2DeviceCodeVerification** を発生させて `user_code` を検証し、デバイスを認可します。
6. 一方、デバイスはトークンエンドポイントを `grant_type=urn:ietf:params:oauth:grant-type:device_code&device_code=...&client_id=...` でポーリングします
7. ユーザーが承認するまでサーバーは `authorization_pending` を返します。承認されると、アクセストークンを返します。

設定

プロパティ	説明
<code>OAuth2Options.DeviceAuthorization.Enabled</code>	デバイス認可エンドポイントを有効にするには True に設定します。デフォルト: False。
<code>OAuth2Options.DeviceAuthorization.URL</code>	デバイス認証エンドポイントの URL パス。デフォルト: <code>/sgc/oauth2/device</code>
<code>OAuth2Options.DeviceAuthorization.VerificationURL</code>	ユーザーがコードを入力する認証ページの URL パス。デフォルト: <code>/sgc/oauth2/device/verify</code>
<code>OAuth2Options.DeviceAuthorization.ExpiresIn</code>	デバイスコードの有効期間（秒）。デフォルト値: 600（10分）。
<code>OAuth2Options.DeviceAuthorization.Interval</code>	デバイスが使用すべき最小ポーリング間隔（秒単位）。デフォルト: 5。

イベント

イベント	説明

<code>OnOAuth2DeviceAuthorization</code>	デバイスがデバイスコードをリクエストしたときに発生します。レスポンスのカスタマイズやリクエストのログ記録を行えます。
<code>OnOAuth2DeviceCodeVerification</code>	ユーザーがデバイス確認ページで確認コードを送信したときに発生します。user_code を検証して、デバイスを承認または拒否してください。
<code>OnOAuth2AfterAccessToken</code>	サーバーがデバイスにアクセストークンを発行した後に発生します。

使用例

```
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);
}
```

ポーリング動作

ユーザーがデバイスをまだ認可していない間、デバイスは設定されたインターバルでトークンエンドポイントをポーリングします。サーバーは以下で応答します。

- authorization_pending -- ユーザーがまだ認証を完了していません。
- slow_down -- デバイスのポーリング頻度が高すぎます。デバイスはインターバルを5秒増やす必要があります。
- expired_token -- デバイスコードが期限切れです。デバイスはフローを再開する必要があります。
- access_denied -- ユーザーが認証を拒否しました。

ユーザーがデバイスを認可すると、次のポーリングでアクセストークンが返されます。

OAuth2 サーバー | トークン失効 (RFC 7009)

概要

トークン失効は、クライアントが以前に発行されたアクセストークンまたはリフレッシュトークンが不要になったことを [TsgcHTTP_OAuth2_Server](#) に通知できる機能です。サーバーはトークンを無効化し、保護されたリソースへのアクセスに使用できなくなります。

これは、ユーザーがログアウトしたとき、アプリケーションがアンインストールされたとき、またはトークンが侵害された可能性があるときに役立ちます。

Endpoint

/sgc/oauth2/revoke に以下のパラメーターで POST します。

パラメータ	説明
<code>token</code>	失効させるトークン（必須）。
<code>token_type_hint</code>	トークンタイプに関するオプションのヒント: <code>access_token</code> または <code>refresh_token</code> 。

動作

[RFC 7009](#) に基づき、サーバーはトークンが見つかったかどうか、または失効に成功したかどうかに関わらず、常に HTTP 200 OK を返します。これにより、トークンの存在が漏洩するのを防ぎます。クライアントはレスポンスステータスを観察することでトークンが有効だったかどうかを判断できません。

リフレッシュトークンが失効すると、関連するアクセストークンも無効になる場合があります（実装依存）。アクセストークンが失効した場合、関連するリフレッシュトークンは明示的に失効させない限り有効のままです。

設定

プロパティ	説明
<code>OAuth2Options.Revocation.Enabled</code>	失効エンドポイントを有効にするにはTrueに設定します。デフォルト: False。
<code>OAuth2Options.Revocation.URL</code>	失効エンドポイントの URL パス。デフォルト: /sgc/oauth2/revoke

イベント

イベント	説明
<code>OnOAuth2AfterRevokeToken</code>	トークン失効の試みの後に発生します。トークン値、token_type_hint、およびトークンが正常に無効化されたかどうかを示す <code>Revoked</code> パラメーターを提供します。失効アクティビティのログ記録に役立ちます。

使用例

```
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;
}
```

クライアントリクエストの使用例

クライアントは失効エンドポイントに POST リクエストを送信してトークンを失効させます。

```
// Client-side: revoke an access token
HTTPClient->Post("https://example.com/sgc/oauth2/revoke",
    "token=eyJhbGciOiI...&token_type_hint=access_token");
```

OAuth2 サーバー | トークンイントロスペクション (RFC 7662)

概要

トークンイントロスペクションにより、リソースサーバーはTsgcHTTP_OAuth2_Serverにクエリして、アクセストークンまたはリフレッシュトークンのアクティブ状態とメタデータを確認できます。これは、リソースサーバーがローカルでデコードせずにトークンを検証する必要がある場合に役立ちます。

Endpoint

次のパラメータで /sgc/oauth2/introspect に POST してください:

パラメータ	説明
<code>token</code>	イントロスペクトするトークン (必須)。
<code>token_type_hint</code>	トークンタイプに関するオプションのヒント: <code>access_token</code> または <code>refresh_token</code> 。

レスポンス

イントロスペクションエンドポイントは以下のフィールドを含む JSON オブジェクトを返します:

フィールド	説明
<code>active</code>	Boolean。トークンが現在アクティブ (有効で、期限切れまたは失効していない) の場合は True。
<code>scope</code>	トークンに関連付けられたスコープ。
<code>client_id</code>	トークンをリクエストしたアプリケーションのクライアント識別子。
<code>token_type</code>	トークンのタイプ (例: Bearer)。
<code>exp</code>	トークンの有効期限 (Unix タイムスタンプ)。
<code>username</code>	トークンに関連付けられたユーザー名 (該当する場合)。

トークンが無効、期限切れ、または失効している場合、応答には{"active": false}のみが含まれます。

設定

プロパティ	説明
<code>OAuth2Options.Introspection.Enabled</code>	introspectionエンドポイントを有効にするにはTrueに設定します。デフォルト: False。

`OAuth2Options.Introspection.URL`

イントロスペクションエンドポイント URL パス。デフォルト: `/sgc/oauth2/introspect`

イベント

イベント	説明
<code>OnOAuth2AfterIntrospectToken</code>	トークンのイントロスペクションリクエストの後に発生します。トークンの値と、デフォルトの検証結果を上書きするために変更できる <code>IsActive</code> パラメータを提供します。カスタム検証ロジックや監査に役立ちます。

使用例

```
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));
}
```

リソースサーバーリクエストの例

リソースサーバーはイントロスペクションエンドポイントに POST リクエストを送信してトークンを検証します:

```
// 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 サーバー | DPoP 検証 (RFC 9449)

概要

DPoP (Demonstrating Proof of Possession) ([RFC 9449](#)) を使用すると、[TsgcHTTP_OAuth2_Server](#) が送信者に制約されたトークンを要求できます。DPoP が有効な場合、サーバーは DPoP 証明 JWT を検証し、クライアントの JWK サムプリントにトークンをバインドし、Bearer の代わりに token_type: DPoP を返します。これにより、アクセストークンは最初に取得したクライアントのみが使用できるようになり、トークンの盗難とリプレイ攻撃を防ぎます。

仕組み

- クライアントはトークンリクエストに署名付き JWT プルーフを含む DPoP ヘッダーを含めます。
- サーバーは証明 JWT を検証します:
 - typ は dpop+jwt である必要があります
 - アルゴリズム (alg) は非対称 (例: ES256、RS256) である必要があります
 - htm (HTTP メソッド) と htu (HTTP URI) はリクエストと一致する必要があります
 - iat (発行時刻) は最近のものである必要があります (鮮度チェック)
 - 署名は証明ヘッダーに埋め込まれた JWK に対して検証されます
- サーバーは JWK サムプリントを抽出し、発行されたトークンにバインドします。
- トークンレスポンスには Bearer の代わりに token_type: DPoP が含まれます。
- 後続のリソースリクエストで、サーバーはプルーフが正しいトークンにバインドされていることを確認するために ath クレーム (アクセストークンハッシュ) を含む DPoP プルーフを検証します。

設定

プロパティ	説明
<code>OAuth2Options.DPoP</code>	Boolean。DPoP バウンドトークン検証を有効にするには True に設定します。デフォルト: False。

署名の検証

サーバーは DPoP 証明ヘッダーに含まれる JWK から EC または RSA 公開鍵を再構築し、JWT 署名を検証します。非対称アルゴリズム (ES256、ES384、ES512、RS256、RS384、RS512、PS256、PS384、PS512) のみが受け入れられます。対称アルゴリズム (HS256 など) は拒否されます。

Nonceのサポート

サーバーはトークンレスポンスに DPoP-Nonce ヘッダーを発行できます。存在する場合、クライアントは後続の DPoP プルーフ JWT の nonce クレームにノンズ値を含める必要があります。これによりリプレイ保護の追加レイヤーが提供されます。

イベント

イベント	説明

OnOAuth2ValidateDPoP

標準の DPoP 検証が完了した後に発火します。追加のクレームの確認やポリシーの施行などのカスタム検証ロジックを使用できます。証明を拒否するには **Valid** を **False** に設定します。

使用例

```
TsgcHTTP_OAuth2_Server *OAuth2Server = new TsgcHTTP_OAuth2_Server(this);
OAuth2Server->OAuth2Options->DPoP = true;
OAuth2Server->Apps->AddApp("MyApp", "http://127.0.0.1:8080",
    "my-client-id", "my-client-secret", 3600, true,
    TsgcOAuth2GrantTypes() << auth2Code);
Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2 = OAuth2Server;
void __fastcall TForm1::OAuth2ServerOAuth2ValidateDPoP(TObject *Sender,
    const UnicodeString DPoPProof, const UnicodeString AccessToken,
    const UnicodeString Method, const UnicodeString URL, bool &Valid)
{
    Log("DPoP proof validated for: " + Method + " " + URL);
    Valid = true;
}
```

無効な証明の拒否

DPoP 証明の検証が失敗すると、サーバーはエラーレスポンスとして HTTP 400 を返します:

```
{"error": "invalid_dpop_proof", "error_description": "..."}

```

拒否の一般的な理由には以下が含まれます。

- DPoP ヘッダーが欠落しているか不正な形式です。
- 無効な JWT 署名です。
- 使用された対称アルゴリズム（非対称アルゴリズムのみが許可されています）。
- htm または htu クレームの不一致。
- 有効期限切れまたは未来の iat クレーム。
- リソースリクエストでの ath（アクセストークンハッシュ）の不一致。
- サーバーが要求する場合に、nonceが無効または欠落しています。

TsgcHTTP_OAuth2_Server_Provider

このコンポーネントを使用すると、外部OAuth2プロバイダー（Azure AD、Google、Facebookなど）を統合できます。

はじめに

このコンポーネントを使用して、外部 OAuth2 プロバイダー（Azure AD、Google、Facebook など）をサーバーコンポーネント（HTTP サーバーなど）に統合できます。これにより、ユーザーが Azure AD の認証情報を使用してログインでき、認証が成功した場合、HTTP サーバーが保護されたリソースへのアクセスを提供できます。

サーバーコンポーネントには `Authorization.OAuth.OAuth2Provider` プロパティがあり、`TsgcHTTP_OAuth2_Server_Provider` のインスタンスを割り当てることができます。認証が有効で `OAuth2Provider` プロパティが OAuth2 プロバイダーサーバーコンポーネントにアタッチされている場合、WebSocket および HTTP リクエストの処理には Cookie/ベアラートークンが必要です。それがない場合、接続は自動的に閉じられます。

```
TsgcHTTP_OAuth2_Server_Provider *OAuth2Provider = new TsgcHTTP_OAuth2_Server_Provider(this);
Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2Provider = OAuth2Provider;
```

OAuth2 プロバイダーの登録

サーバーが起動する前に、認証に使用する OAuth2 プロバイダーを設定する必要があります。**RegisterProvider** メソッドを使用して OAuth2 プロバイダーを設定します。このメソッドには以下のパラメータがあります：

- **Name**：プロバイダーの名前です。任意の名前を付けられます。後でプロバイダーを識別するためのものです。
- **ClientId**：パブリッククライアント ID です。この値は OAuth2 プロバイダーによって提供されます。
- **ClientSecret**：プライベートクライアントシークレット（機密に保つ必要があります）。この値は OAuth2 プロバイダーによって提供されます。
- **AuthorizeURL**：OAuth2 クライアントがログインのためにリダイレクトする URL（接続は Web ブラウザを使用）です。
- **TokenURL**：成功した認証後に提供されたトークンを検証するためにサーバーが使用する URL です（接続はサーバー間です）。
- **Scope**：スコープの値です。
- **URL**：認証 URL へのリダイレクトに使用される HTTP サーバーの URL です。
- **CallbackURL**：認証の成功後に OAuth2 サーバーから送信された応答を処理する OAuth2 プロバイダーで設定された URL です。

例：Azure AD を設定するには、OAuth2 URL に追加されるテナント ID、ClientId、ClientSecret、Scope、および CallbackURL が必要です。

```
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'
);
```

既存のプロバイダーを削除するには、**UnRegisterProvider** メソッドを使用してください。

リファレンス

- プロパティ — コンポーネントで利用可能な2つのプロパティ。
- メソッド — パブリックメソッドが 15 個あります。
- イベント — 処理可能な4個のイベント。

ガイド

- **QuickStart**
 - [OAuth2 プロバイダー Azure AD](#)
- **認証**
 - [OAuth2 プロバイダープライベートエンドポイント](#)
 - [OAuth2 プロバイダー認証](#)
 - [OAuth2 プロバイダーリクエスト](#)

OAuth2 Provider | Azure AD

Azure AD は以下の OAuth2 認可 URL を使用します。

Authorization : <https://login.microsoftonline.com/<tenant-id>/oauth2/v2.0/authorize>

Token: <https://login.microsoftonline.com/<tenant-id>/oauth2/v2.0/token>

<tenant-id> は独自の値に置き換える必要があります。

OAuth2 設定を作成するときは、サーバー コールバック URL を設定する必要があります。この URL は、承認が成功した後に Azure がサーバーに応答を送信するために使用されます。

例 : Azure AD プロバイダーを登録する簡単な例を以下に示します。

Azure ADが提供する値

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 AD の登録方法

```
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 プロバイダー | プライベートエンドポイント

サーバーが HTTP リクエストを受信するたびに、**OnOAuth2IsPrivateEndpoint** イベントが呼び出され、エンドポイントがプライベートかどうかを尋ねます。デフォルトではプライベートではありません。

```
void OnOAuth2IsPrivateEndpoint(TObject *Sender, const string aEndpoint, ref bool IsPrivate)
{
    if (aEndpoint == "/private")
    {
        IsPrivate = True;
    }
}
```

OAuth2 Provider | Authentication

OAuth2 プロバイダーサーバーコンポーネントを使用すると、外部 OAuth2 プロバイダー（Azure AD や Google など）を使用して認証し、サーバーの保護されたリソースにアクセスできます。**例:** Azure 認証情報を使用してユーザーがログインできるように HTTP サーバーを設定できます。ログインが成功すると、それらのユーザーはサーバーの保護されたリソースにアクセスできるようになります。

認証プロセスはサーバー側で行われ、OAuth2 トークンはクライアントと共有されません。つまり、ユーザーが Azure などを使用してログインした場合、認証が成功すると、Azure はユーザープロファイルに関する情報（スコープに応じて）、メールなどを Azure サーバーからリクエストするためのアクセストークンを返します。このアクセストークンはクライアント（例えば Web ブラウザ）には**共有されません**。クライアントにアクセストークンを返す代わりに、サーバーは内部でアクセストークンにリンクされたランダム ID を作成します。クライアント（Web ブラウザ）が OAuth2 サーバーへの呼び出しを行うたびに、この公開 ID を使用し、サーバーはこの ID を使用して HTTP リクエストをプロキシするための OAuth2 アクセストークンを取得します。

OAuth2 認証の動作例を以下に示します。この例では、次のリンクで説明されている Azure AD 設定を使用します：[OAuth2 プロバイダー Azure AD](#)。

サーバーを起動する

サーバーは localhost とポート 443 でリッスンを開始します。sgcWebSockets HTTP サーバーは OAuth2 サーバープロバイダーコンポーネントにリンクされており、Authentication プロパティが有効になっています。サーバーが起動される前に、Azure OAuth2 プロバイダーは以下のメソッド呼び出しを使用して登録されます。

```
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'  
);
```

ユーザーログイン

ユーザーが新しいWebブラウザを開き、'/login' エンドポイントにアクセスします。サーバーは、Azure プロバイダーを使用してログインするために「/login」エンドポイントが使用されていることを検出し、以下にリダイレクトします。

<https://login.microsoftonline.com/a0ca2055-5dd1-467f-bf13-291f6fd715c6/oauth2/v2.0/authorize>

OAuth2 認証フローが開始されます。

OAuth2 Authentication

ユーザーは OAuth2 サーバー認証エンドポイントにリダイレクトされます。ここで資格情報を使用してログインし、OAuth2 アプリケーションの利用規約に同意する必要があります。

認可が成功すると、Azure ADはCodeをURLに送信します

<https://localhost/callback>

OAuth2 Codeを検証する

現在、サーバーは Azure からコードを受け取り、このトークンが正しいことを検証するために Azure への内部接続（サーバーからサーバーへ）を行います（誰かがサーバーをハッキングしようとしていないことを確認するため）。

サーバーが接続する先

`https://login.microsoftonline.com/a0ca2055-5dd1-467f-bf13-291f6fd715c6/oauth2/v2.0/token`

受信したコードやclientsecretなどのパラメータを渡し、検証が成功すると、Azureはアクセストークンを返します。これは、プロフィールやメールの読み取りなど、Azure Protected Resourcesへのアクセスに使用できます。

アクセストークン成功

サーバーが正常な AccessToken を受信すると、イベント **OnOAuth2ProviderTokenValid** が呼び出されます。ここでパラメータークラス `TsgcHTTPOAuth2ProviderToken` にアクセスして AccessToken の保存方法（保存する場合）を設定できます

AccessToken: Azure によって返された OAuth2 トークンです。

ID: Cookieとして保存されるパブリック識別子です。

このイベントでは、認証成功後に何をするかを設定できます。例: ユーザーをプライベート URL にリダイレクトしたい場合は、以下を使用します。

`Response.Redirect.URL := 'https://localhost/private';`

Azure へのリクエスト送信

これで、Cookie として保存された公開 ID を使用して Azure サーバーにリクエストを送信できます。例: プロファイルデータを読み取りたい場合は、以下のメソッドを使用します。

`Get('ID', 'https://graph.microsoft.com/v1.0/me');`

ID はパブリック ID 識別子です。

OAuth2 プロバイダー | リクエスト

認証が成功したら、パブリック ID トークンを Cookie として使用して OAuth2 保護サーバーにリクエストを送信できます。

OAuth2 プロバイダーサーバーコンポーネントには、HTTP リクエストを送信するためのいくつかのメソッドがあります：GET、POST、DELETE...

Indy サーバーコンポーネントを使用している場合は、トークンをパラメーターとして渡すか、RequestInfo クラスを渡すことができます。

```
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）は通常、ヘッダー + ペイロード + 署名で構成されます。

ヘッダー

JWT に関する メタデータ情報が含まれています。

- **alg**: トークンの署名に使用されるアルゴリズム。
- **typ**: トークンのタイプ。常に JWT。

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

他にもヘッダーが存在する場合がありますが、上記のヘッダーは 常に存在します。

ペイロード

ペイロードには JWT のクレームが含まれています。標準ヘッダーは以下のとおりです:

- **iss**: JWT を生成・発行するエンティティである発行者。
- **sub**: サブジェクト、このトークンによって識別されるエンティティ。
- **aud**: オーディエンス。このJWTの対象オーディエンス。
- **exp**: 有効期限。トークンを受け入れてはならない日時のUNIXタイムスタンプ。
- **iat**: 発行日時。トークンが発行された日付を指定します。

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

署名

署名は、エンコードされたヘッダー、エンコードされたペイロード、シークレット、および暗号アルゴリズムを使用して作成されます。

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MzkwMj0
```

```
DIyfq.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

サポートされているアルゴリズム

次のアルゴリズムはクライアントとサーバーの両方の JWT コンポーネントでサポートされています。

- HS256
- HS384
- HS512
- RS256
- RS384
- RS512
- ES256
- ES384
- ES512

JWTの署名と検証には、**OpenSSL**ライブラリが必要です。

コンポーネント

- **TsgcHTTP_JWT_Client** : JWT をエンコードおよび署名し、**HTTP** および **WebSocket** プロトコルの Authorization ヘッダーとして送信できる JWT クライアント。
- **TsgcHTTP_JWT_Server**: **HTTP** および **WebSocket** プロトコルの Authorization ヘッダーとして受信した JWT をデコードおよび検証できる JWT サーバー。

* JWT コンポーネントには少なくとも *Indy* バージョン 10.6.0.5169 または *sgcWebSockets Enterprise* エディションが必要です。

TsgcHTTP_JWT_Client

JWT クライアント使用時に使用する OpenSSL ライブラリを設定します。

はじめに

TsgcHTTP_JWT_Clientコンポーネントを使用すると、JWTトークンをエンコードして署名できます。[WebSocket クライアント](#)または[HTTP/2クライアント](#)にアタッチすることで、トークンが自動的にAuthorization Bearer Token ヘッダーとして送信されます。

設定

JWTOptions プロパティでJWT値を設定できます。**Header** と **Payload** という2つのメインプロパティがあります。必要なプロパティごとに値を設定してください。

署名がプライベートキー（RS および ES アルゴリズム）を使用して暗号化されている場合は、アルゴリズムの **PrivateKey** プロパティに値を設定してください。

シグネチャが Secret を使用して暗号化されている場合 (HS アルゴリズム)、Algorithm の **Secret** プロパティに値を設定します。

OpenSSL オプション

JWT クライアント使用時に使用する OpenSSL ライブラリを設定します。

OpenSSL_Options: OpenSSL ライブラリの設定。

APIVersion: 使用する OpenSSL API を定義できます。

oslAPI_1_0: API 1.0 OpenSSL を使用します。Indy で最後にサポートされたバージョンです。

oslAPI_1_1: API 1.1 OpenSSL を使用します。カスタム Indy ライブラリが必要で、OpenSSL 1.1.1 ライブラリ (TLS 1.3 サポートあり) を使用できます。

oslAPI_3_0: API 3.0 OpenSSL を使用します。カスタム Indy ライブラリが必要で、OpenSSL 3.0.0 ライブラリ (TLS 1.3 サポートあり) を使用できます。

LibPath: opensslライブラリの場所を設定できます

oslpNone: これはデフォルトです。opensslライブラリは、バイナリと同じフォルダー、または既知のパスにある必要があります。

oslpDefaultFolder: すべてのIDEパーソナリティに対してライブラリが配置されるべき opensslパスを自動的に設定します。

oslpCustomFolder: このオプションが選択されている場合、プロパティ LibPathCustom にフルパスを定義します。

LibPathCustom: LibPath = oslpCustomFolder の場合、openssl ライブラリが格納されているフォルダーの完全パスをここで定義します。

UnixSymLinks: Unix システムでシンボリックリンクの読み込みを有効または無効にします（デフォルトでは有効です。OSX64 は除きます）。

oslsSymLinksDefault: デフォルトでは有効です。ただし OSX64 では無効です（MacOS Monterey 以降、バージョンなしでライブラリを読み込もうとすると失敗するため）。

oslsSymLinksLoadFirst: SymLinksを読み込み、バージョンライブラリを読み込む前に行います。

oslsSymLinksLoad: バージョンライブラリの読み込みを試みた後にシンボリックリンクを読み込みます。

oslsSymLinksDontLoad: SymLinksを読み込みません。

カスタムヘッダー

Header プロパティと Payload プロパティには JWT を生成するために使用される最も一般的なヘッダーが含まれていますが、**AddKeyValue** メソッドを呼び出してキーと値をパラメータとして渡すことでさらにヘッダーを追加できます。

例：JWT ヘッダーに名前の新しいレコードを追加したい場合は、次のメソッドを使用します。

```
Header->AddKeyValue("name", "John Smith");
```

プロパティを設定した後、JWTを生成するには、メソッド**Sign**を呼び出すだけで、JWTの値が返されます。

WebSocket クライアントと JWT

[TsgcWebSocketClient](#) では、WebSocketサーバーに接続する際にJWTを使用できます。新しいJWTクライアントを作成し、**Authentication.Token.JWT** プロパティに割り当ててください。

```
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 クライアントと JWT

[TsgcHTTP2Client](#) および[TsgcHTTP1Client](#)では、HTTP/2サーバーに接続する際にJWTを使用できます。新しいJWTクライアントを作成し、**Authentication.Token.JWT** プロパティに割り当ててください。

```
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");
```

有効期限

認証トークンは HTTP クライアントを使用して HTTP リクエストを送信するたびに**再作成**することも、**期限が切れるまで何度も再利用**することもできます。

例：トークンを使用して Apple APNs を呼び出す場合、トークンは少なくとも 20 分間再利用され、最大 1 時間使用されることが必要です。プロパティ **RefreshTokenAfter** を使用して、トークンが期限切れになるまでの秒数を設定します（例えば 30 分後）。

```
RefreshTokenAfter = 60 * 40.
```

JWT 署名の作成

WebSocketまたはHTTPプロトコルを使用しないアプリケーション、あるいはサードパーティのアプリケーションのコンポーネントを使用してJWT Tokenのみが必要な場合に使用するために、**JWT署名を手動で作成**できます。

JWT署名を取得するには、新しいJWT Clientのインスタンスを作成し、プロパティを手動で入力します。すべてのプロパティが設定されたら、**Sign**メソッドを呼び出すとJWTトークンが返されます。

```
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());
```

リファレンス

- プロパティ — コンポーネントで利用可能な2個のプロパティ。
- メソッド — 2個のパブリックメソッド。
- イベント — 処理できるイベント 0 件。

TsgcHTTP_JWT_Server

TsgcHTTP_JWT_Server コンポーネント。

はじめに

TsgcHTTP_JWT_Server コンポーネントを使用すると、WebSocket プロトコルを使用する際は **WebSocket** ハンドシェイクで、HTTP プロトコルを使用する際は **HTTP** ヘッダーとして受信した JWT トークンをデコードおよび検証できます。

設定

コンポーネントの **JWTOptions** プロパティで次のプロパティを設定できます：

公開鍵（RS および ES アルゴリズム）を使用して署名を検証する場合は、アルゴリズムの **PublicKey** プロパティに値を設定してください。

シークレット（HS アルゴリズム）を使用して署名を検証する場合は、アルゴリズムの **Secret** プロパティに値を設定します。

JWT トークンを検証するには、**TsgcHTTP_JWT_Server** インスタンスをアタッチするだけで、WebSocket/HTTP サーバーの **Authentication.JWT.JWT** プロパティに設定します。

```
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 プロパティを使用してJWTのペイロードに対していくつかのチェックを有効にできます。デフォルトでは発行日が有効かどうかを確認します。

リファレンス

- プロパティ — コンポーネントで使用可能な 2 つのプロパティ。
- メソッド — 3 つのパブリックメソッド。
- イベント — 処理できるイベント 7 件。

WebAuthn

WebAuthn (Web Authentication) は、Web上でセキュアかつパスワードレス認証を可能にするために World Wide Web Consortium (W3C) と FIDO Alliance によって開発された Web 標準です。これはより広範な FIDO2 フレームワークの一部であり、フィッシング、クレデンシャルスタッフィング、その他の攻撃に対して脆弱であることが多い従来のパスワードへの依存を減らすことを目的としています。

WebAuthn を使用すると、ユーザーは公開鍵暗号を使用して認証できます。ユーザー名とパスワードの代わりに、ユーザーはウェブアプリケーション (Relying Party) に一意の公開鍵・秘密鍵ペアを登録します。秘密鍵はハードウェアセキュリティキー、スマートフォン、または組み込みの生体認証デバイスなどの認証器に安全に保存され、公開鍵はサーバーに保存されます。

認証中、サーバーはユーザーの秘密鍵で署名される必要があるチャレンジを発行します。署名されたチャレンジが返され、保存された公開鍵を使用して検証されます。これにより、レスポンスの整合性と発信元の両方が保証されます。このアプローチにより、資格情報の傍受や再利用を防止します。

WebAuthnは、さまざまな認証器とデバイスをサポートしており、開発者とユーザーの両方にとって柔軟です。また、PINやバイオメトリクスなどの他の要素と組み合わせると、多要素認証 (MFA) が可能になり、使いやすさを犠牲にすることなくセキュリティが大幅に向上します。

サーバー

- **TsgcWSAPIServer_WebAuthn:** このコンポーネントは、WebAuthn Relying Partyサーバーを実装するためのシンプルかつ強力なソリューションを提供し、Webアプリケーションでパスワードレス認証を可能にします。サーバーは、Conformance Self-Validation Testingツールを使用して完全な**FIDO Conformance Test**に合格しています。

クライアント

- **JavaScript クライアント:** 登録と認証の際に、ユーザー、ブラウザ、および認証器の橋渡し役を担います。

TsgcWSAPIServer_WebAuthn

OpenSSL 3 のみ。

はじめに

TsgcWSAPIServer_WebAuthn コンポーネントは、ウェブアプリケーションでパスワードレス認証を実現するために WebAuthn Relying Party サーバーを実装するシンプルで強力なソリューションを提供します。WebAuthn アプリケーションは、サーバー側の登録と認証を処理する WebAuthn サーバーと、通常は JavaScript アプリケーションであるクライアント側アプリケーションで構成されます。

WebAuthn はセキュアな接続 (SSL/TLS) を必要とするため、[OpenSSL](#) ライブラリをサーバーにデプロイして設定する必要があります。

OpenSSL 3.0.0+ APIのみがサポートされています。 そのため、以前の OpenSSL バージョンは動作しない可能性があります。

設定

TsgcWSAPIServer_WebAuthn は、Server プロパティを使用して HTTP サーバー ([TsgcWebSocketHTTP_Server](#) または [TsgcWebSocketServer_HTTPAPI](#)) にアタッチする必要があります。登録と認証オプションを処理するサーバーエンドポイント、サポートされているアルゴリズム、オリジンなどの WebAuthn オプションを設定できます。

エンドポイントオプション

WebAuthn を認証子として使用するための HTTP/JavaScript リクエストを処理するサーバーエンドポイントをここで設定できます。コンポーネントにはデフォルトのエンドポイントが設定済みですが、必要に応じてすべて変更できます。

- **AuthenticationOptions:** デフォルトは /sgcWebAuthn/Authentication/Options
- **AuthenticationVerify:** デフォルトは /sgcWebAuthn/Authentication/Verify です
- **RegistrationOptions:** デフォルトは /sgcWebAuthn/Registration/Options です
- **RegistrationVerify:** デフォルトでは /sgcWebAuthn/Registration/Verify です
- **Webauthn:** デフォルトで使用する JavaScript ライブラリを含みます。このプロパティを無効にして独自の webauthn ライブラリを使用することもできます。
- **Test:** デフォルトでは無効です。WebAuthn 機能をテストする場合にのみ使用してください。

使用例: サーバーがドメイン `www.test.com` でリッスンしている場合、認証オプションへのデフォルトのリクエスト URL は `http://www.test.com/sgcWebAuthn/Authentication/Options` になります。

WebAuthn オプション

このプロパティでは WebAuthn サーバーコンポーネントの主要なオプションを設定できます。

- **RelyingParty:** 必須プロパティで、サーバーの DNS 名を定義する必要があります。例: サーバーがドメイン `www.test.com` で動作している場合、このプロパティを「`www.test.com`」に設定します。

WebAuthn は同一オリジンポリシーの制約を適用するためにオリジンを使用します。これはフィッシングやクロスサイト攻撃を防ぐために不可欠です。WebAuthn の登録と認証プロセス中、オリジンはブラウザーと認証器によって厳密に検証されます。

- **Origins:** リクエストが異なるオリジンから来る可能性がある場合は、Origin プロパティを使用して追加のオリジンを設定します。例: リクエストが login.test.com と www.test.co.uk から来る可能性がある場合は、Origins プロパティに https://login.test.com と https://www.test.co.uk の値を設定します。
- **TopOrigins:** 通常、WebAuthn は呼び出しフレームのオリジン (navigator.credentials.create() または navigator.credentials.get() を呼び出すもの) に依存します。しかし、Web ページは異なるオリジンから来る可能性のある iframe に埋め込まれることがあります。これにより、悪用やクリックジャッキング型の攻撃の可能性が生じます。これを軽減するために、WebAuthn レベル 2 の仕様では TopOrigins を定義できる TopOrigin を導入しています。

WebAuthn では、crossOrigin はトップレベルのブラウジングコンテキストとは異なるオリジンから埋め込まれた iframe などのクロスオリジンコンテキストから WebAuthn 操作が実行されているかどうかを示すブール値パラメーターです。

このパラメータは、埋め込み環境での認証リクエストをブラウザおよびオーセンティケーターが安全に処理できるよう支援するために導入されました。これは最新のWebアプリケーションにおける一般的なシナリオです。

- **AllowCrossOrigins:** true の場合、クロスオリジン iframe (例: https://app.example.org のページに埋め込まれた https://auth.example.com の iframe) からのリクエストが許可されることを示します。デフォルトでは無効です。

WebAuthn は、公開鍵資格情報の生成と検証のためにさまざまな**暗号アルゴリズム**をサポートしています。これらのアルゴリズムは、資格情報の登録 (navigator.credentials.create()) と認証 (navigator.credentials.get()) の際に使用され、非対称鍵ペアを用いてチャレンジの安全な署名と検証を保証します。サーバーはデフォルトで最も一般的なアルゴリズムである **ES256** と **RS256** で設定されています。**Algorithms** プロパティからサポートするアルゴリズムをいつでも変更できます。以下のアルゴリズムがサポートされています。

- ES256
- ES384
- ES512
- RS256
- RS384
- RS512
- PS256
- PS384
- PS512
- RS1
- EdDSA

WebAuthn において、**証明書はオプションのメカニズム**で、資格情報の作成時に認証器（デバイスやセキュリティキーなど）がメーカー、モデル、セキュリティ特性に関する情報を提供できるようにします。この情報は依頼当事者（RP）が認証器を信頼するかどうかを判断するのに役立ちます。

異なるアテステーション形式は、このデータの構造化と検証方法を定義します。一般的に使用される 3 つの形式は android-key、packed、およびその他 (fido-u2f、apple、none) です。デフォルトでは、すべてのアテステーション形式が有効になっています。サポートされているアテステーション形式のリストは以下を参照してください:

- **NoneAttestation:** この場合、アテステーションデータは返されません。デバイス識別子の公開を避けることでユーザーのプライバシーを優先します。デバイスの出所を気にしないアプリケーションでよく使われます。
- **PackedAttestation:** 多くの認証器が使用する柔軟でコンパクトなフォーマットです。認証器はアテステーション証明書と署名を返します。**完全アテステーション:** ベンダー提供のキーと証明書で署名されたもの、または **自己アテステーション:** 資格情報の秘密鍵を使用して署名されたものがあります。さまざまなプラットフォーム (例: YubiKey、Windows Hello) で最も広く使用されています。
- **TPMAttestation:** トラステッドプラットフォームモジュール (TPM) を搭載したデバイスで使用されます。アテステーションは TPM のキーを使用して署名され、証明書チェーンが含まれます。TPM チップを搭載したエンタープライズデスクトップ/ノートパソコン (例: Windows マシン) で使用されます。
- **AndroidKeyAttestation:** Android Keystore を使用する Android デバイスで使用されます。キーはハードウェアで生成され、証明書にはデバイスメーカーが発行した証明書チェーンによって署名された情報が含まれます。ハードウェアバックのキーストア (TEE または StrongBox) を持つ Android 携帯電話で使用されます。

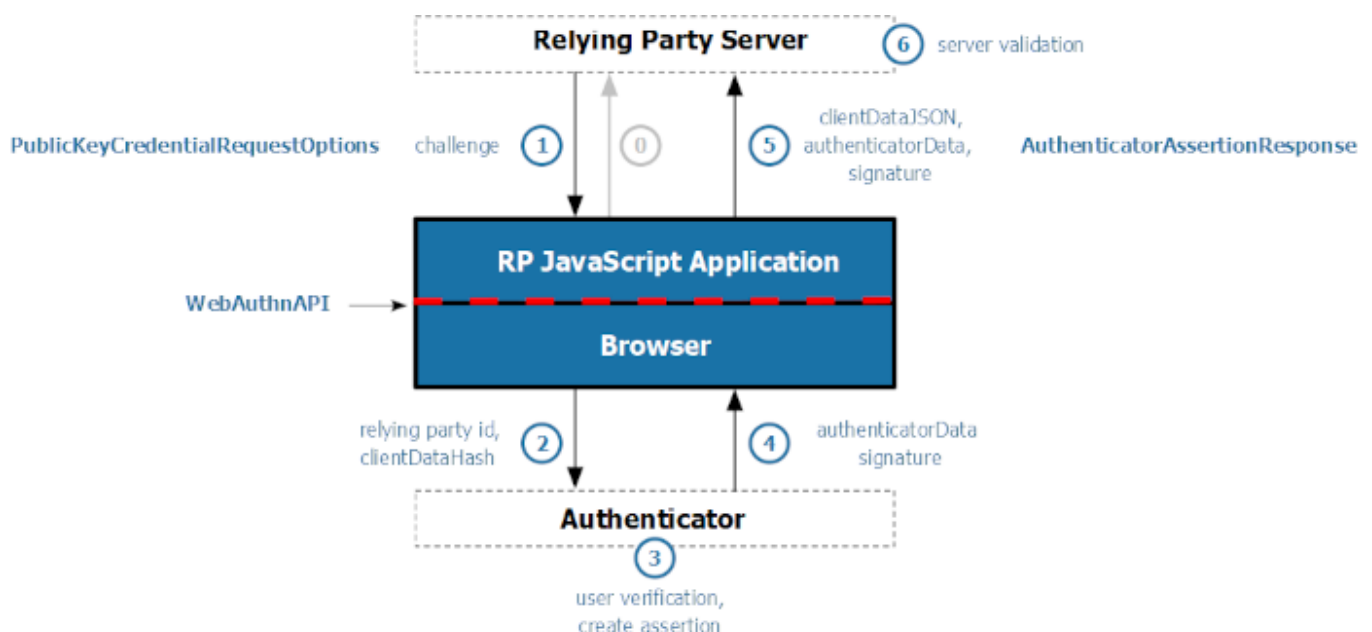
- **AppleAttestation:** Touch ID や Face ID などの Apple プラットフォーム認証器で使用されます。アテステーションは Apple の内部 API によって生成され、特別な証明書形式が含まれます。Apple バイオメトリクスを使用した Safari で使用されます。
- **FidoU2FAttestation:** FIDO U2F 認証器が使用するレガシーアテステーションフォーマット。U2F 互換の証明書と署名を返します。FIDO U2F をサポートする古いセキュリティキー（初期の YubiKey など）で使用されます。

WebAuthn APIにおいて、**AllowCredentials**は認証プロセス中（navigator.credentials.get()経由）に使用されるオプションフィールドです。これは、特定のRelying Party（RP）に対して**ユーザーの認証を許可される資格情報ID**のリストを指定します。このメカニズムにより、RPはログイン試行に対してどの資格情報を有効と見なすかを制御できます。credentialsプロパティには次のフィールドがあります：

- **AllowCredentials**：有効にすると（デフォルトは false）、ユーザーの認証が許可されたクレデンシャル ID のリストを指定します
- **ExcludeCredentials**：ユーザー名を指定すると、サーバーコンポーネントに既に保存されているすべての既存の認証情報を表示します。
- **Limit**：ExcludeCredentials が true の場合に送信される認証情報の最大数。

WebAuthn プロトコル

- **WebAuthn 登録**：サーバーはチャレンジを生成してクライアントに送信し、クライアントは認証器（セキュリティキーや生体認証デバイスなど）を使用して鍵ペアを作成します。公開鍵は送り返され、将来の認証のためにサーバーに保存されます。登録フローのイベントの詳細については以下を参照してください：
 - [WebAuthn 登録リクエスト](#)
 - [WebAuthn 登録レスポンス](#)
 - [WebAuthn 登録結果](#)
- **WebAuthn 認証**：サーバーはクライアントにチャレンジを送信し、クライアントはオーセンティケーターに保存されている以前に登録した秘密鍵を使用してそれに署名します。署名されたレスポンスは、サーバーが保存している公開鍵を使用して検証され、ユーザーの身元が確認されます。認証フローのイベントに関する詳細は以下をご覧ください：
 - [WebAuthn 認証リクエスト](#)
 - [WebAuthn 認証レスポンス](#)
 - [WebAuthn 認証結果](#)
- **MDS**：FIDO Alliance Metadata Service (MDS)は、認証器のアテステーションを検証し、デバイスモデルの真正性を証明するためにリライティングパーティが使用するMetadata Statementの集中リポジトリです。



- **認可:** クライアントは認証フロー中にサーバーからベアータークンをリクエストできます。このトークンは後でパスキーを使用してログインせずに新しい WebSocket または HTTP 接続を開くために使用できます。
 - [WebAuthn 認可 WebSocket](#)
 - [WebAuthn 認可 HTTP](#)

リファレンス

- プロパティ — コンポーネントで利用可能な4つのプロパティ。
- メソッド — 8個のパブリックメソッド。
- イベント — 処理できる15個のイベント。

WebAuthn | 登録

WebAuthn 登録にはクライアント（ブラウザ）、**認証器**（セキュリティデバイス）、および**リライティングパーティ**（RP; TsgcWSAPIServer_WebAuthn Delphi/CBuilderサーバー）が関与します。

登録オプション

新しいユーザー資格情報の作成には通常、クライアントが TsgcWSAPIServer_WebAuthn サーバーで設定された**登録オプション**エンドポイントへの新しい HTTP リクエストを使用して登録フローを開始する必要があります。デフォルトのエンドポイントは /sgcWebAuthn/Registration/Options です。サーバーがドメイン <https://www.test.com> でリスニングしている場合、クライアントは URL <https://www.test.com/sgcWebAuthn/Registration/Options> に新しいリクエストを行う必要があります。

クライアントは、以下の JSON をペイロードとして（ユーザー名として test を使用して）新しいリクエストを送信します。

```
{"username": "test", "algorithms": []}
```

サーバーは要求を読み取り、新しいチャレンジを含む応答を返します。

```
{
  "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
  }
}
```

応答は以下のデータを返します。

- **challenge:** base64 エンコードされたランダムなバイナリデータ。
- **rp:** リライティングパーティ情報（あなたの Web アプリ/サイト）。

- **user** : ユーザー識別子と表示名。
- **pubKeyCredParams**: 許可されるアルゴリズム (例: ES256、RS256)。
- **authenticatorSelection**: オーセンティケーターのタイプ (platform または cross-platform、ユーザー検証オプション) に関する設定です。
- **timeout**: 推奨タイムアウト値。
- **attestation**: アテステーション伝達設定 ("none"、"direct"、"indirect") 。

登録の確認

クライアントがサーバーからレスポンスを受け取ると、レスポンスを読み取り、オーセンティケーターはクライアントのウェブブラウザに暗号化データを返します。クライアントは次に TsgcWSAPIServer_WebAuthn サーバーで設定された登録検証エンドポイント (デフォルトは /sgcWebAuthn/Registration/Verify) に以下のデータを含む新しい HTTP リクエストを送信します。

そのため、サーバーがドメイン <https://www.test.com> でリスニングしている場合、クライアントは URL <https://www.test.com/sgcWebAuthn/Registration/Verify> に新しいリクエストを行う必要があります。

認証者はブラウザの JavaScript に以下で応答します:

- **id**: Credential ID (認証器が生成したキーペアの一意のID) 。
- **rawId**: Credential ID のバイナリ形式。
- **type**: 認証情報の種類 ("public-key") 。
- **response** : attestationObject と clientDataJSON を含みます。

以下に使用例を示します。

```
{
  "id": "yeA4BVRlrAfLG-KzqsL_rll4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
  "rawId": "yeA4BVRlrAfLG-KzqsL_rll4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
  "response": {
    "attestationObject": "o2NmbXRmcGFja2VkZ2F0dFN0....",
    "clientDataJSON": "eyJ0eXBlljoid2ViYXV0aG4uY3Jl....",
    "transports": [
      "nfc",
      "usb"
    ],
    "publicKeyAlgorithm": -7,
    "publicKey": "MFkwEwYHKoZIzj0CAQYIKoZIzj....",
    "authenticatorData": "SZYN5YgOjGh0NBcPZHgW4_k...."
  },
  "type": "public-key",
  "clientExtensionResults": {
    "credProps": {
      "rk": true
    }
  },
  "authenticatorAttachment": "cross-platform"
}
```

サーバーはクライアントからの JSON リクエストを読み取り、公開鍵と資格情報 ID をデコード、検証、保存します。

attestationObject と clientDataJSON をデコードします:

- attestationObject から公開鍵を抽出します。
- clientDataJSON のチャレンジがサーバーから送信された元のものと同じであることを検証します。
- 適切なオリジン検証を確保してください (myapp.example.com) 。

資格情報の保存:

- 検証が成功すると、**OnWebAuthnRegistrationSuccessful** イベントが呼び出されるため、将来の認証のために認証情報 ID と抽出された公開鍵をデータベースに安全に保存できます。
- エラーが発生した場合は、イベント **OnWebAuthnRegistrationError** が呼び出され、エラーの詳細を取得できます。

登録フロー

以下のリンクを使用して登録フロープロセスの詳細を確認します:

- [WebAuthn 登録リクエスト](#)
- [WebAuthn 登録レスポンス](#)
- [WebAuthn 登録結果](#)

WebAuthn Registration | Request

この登録オプションリクエストは、WebAuthn 登録を安全にブートストラップするために必要です。サーバーに以下を要求します。

- チャレンジを生成し、
- ユーザーIDを検索またはプロビジョニングします。
- 有効なPublicKeyCredentialCreationOptionsオブジェクトを返します (RP、ユーザー、アルゴリズムなどを含む)。

ブラウザはセキュアな WebAuthn 資格情報の登録を開始するためにサーバーから資格情報作成オプションを取得する必要があります。これは通常ブラウザからサーバーエンドポイント (/sgcWebAuthn/Registration/Options など) への HTTPS POST リクエストで実行され、チャレンジ、RP ID、ユーザー情報などのパラメータを準備します。

リクエストボディにはユーザーを識別し、場合によっては設定の優先事項を提供するデータが含まれています。通常は次のような形式です:

```
{
  "username": "alice@example.com",
  "displayName": "Alice Smith",
  "authenticatorSelection": {
    "authenticatorAttachment": "platform", // オプション: platform または cross-platform
    "userVerification": "preferred" // オプション: required | preferred | discouraged
  },
  "attestation": "none", // または "direct", "indirect"
  "residentKey": "discouraged" // オプション
}
```

WebAuthn サーバーコンポーネントは、**EndpointOptions.RegistrationOptions** プロパティに設定されたエンドポイントで、PublicKeyCredentialCreationOptions を取得するブラウザの初期リクエストをリッスンします。サーバーが新しい HTTP リクエストを受信すると、**OnWebAuthnRegistrationOptionsRequest** イベントが呼び出され、クライアントから送信されたリクエストにアクセスして、Accept パラメータを False に設定することでリクエストをキャンセルできます。

```
void __fastcall TForm1::OnWebAuthnRegistrationOptionsRequest(TObject *Sender, TsgcWebAuthn_RegistrationOptions_Re
{
    if (aRequest->Username == "anonymous")
        Accept = false;
}
```

WebAuthn 登録 | レスポンス

サーバーはクライアントの登録オプションリクエスト (例: POST /sgcWebAuthn/Registration/Options) に対して、バイナリフィールドを base64url エンコードした後、以下のような JSON ペイロードで応答します:

```
{
  "publicKey": {
    "rp": {
      "name": "esegece software",
      "id": "esegece.com"
    },
    "user": {
      "id": "c3ViamVjdC1pZA", // base64url でエンコードされた ArrayBuffer (ユーザーハンドル)
      "name": "webauthn@esegece.com",
      "displayName": "Delphi Developer"
    },
    "challenge": "Xz8x2K6nY3gZ...", // base64url でエンコードされたチャレンジ
    "pubKeyCredParams": [
      { "type": "public-key", "alg": -7 }, // ES256
      { "type": "public-key", "alg": -257 } // RS256
    ],
    "timeout": 60000,
    "attestation": "none",
    "authenticatorSelection": {
      "authenticatorAttachment": "platform",
      "userVerification": "preferred",
      "residentKey": "discouraged"
    }
  }
}
```

フィールドの説明を以下に示します:

- **rp:** Relying Party情報:
 - **name:** 分かりやすい表示名 (例: "Example Inc.")。
 - **id:** Relying Party ID (サイトのオリジンと一致するか、登録可能なサフィックスである必要があります)。
- **user:** ユーザーアカウントを識別する情報:
 - **id:** 一意で安定したバイト配列 (例: ユーザー UUID)。
 - **name:** 一意のユーザー識別子 (ユーザー名またはメール)。
 - **displayName:** ユーザー向けの表示名。
- **challenge:** 暗号的にランダムなノンス (16~64 バイト)、base64url エンコード。
- **pubKeyCredParams:** RP が受け入れるアルゴリズム:
 - **共通:** -7 (ES256), -257 (RS256)
- **timeout:** 作成操作のオプションのタイムアウト (ミリ秒単位)。
- **attestation:** "none"、"indirect"、"direct" のいずれか。
- **authenticatorSelection:**
 - **authenticatorAttachment:** "platform" (組み込み) または "cross-platform" (例: YubiKey)。
 - **userVerification:** "required"、"preferred"、または "discouraged"。
 - **residentKey:** "required"、"preferred"、または "discouraged"。

クライアントにレスポンスを送信する前に、**OnWebAuthnRegistrationOptionsResponse** イベントが呼び出され、レスポンスをカスタマイズできます。

```
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 登録 | 結果

目標は、クライアントから返されたデータの信頼性と整合性を確認し、クレデンシャルが期待されるユーザーにバインドされていることを確認し、将来の認証のために公開鍵クレデンシャルを安全に登録することです。

サーバーは、以下の手順に従ってクライアントの応答を検証する必要があります：

- クライアント応答の解析: クライアントはこのような応答をサーバーに送信します。

```
{
  "id": "base64url-encoded credential ID",
  "rawId": "base64url-encoded ID bytes",
  "response": {
    "clientDataJSON": "base64url",
    "attestationObject": "base64url"
  },
  "type": "public-key"
}
```

- サーバーは clientDataJSON と attestationObject を検証します。
- Attestation Statement を検証します（定義されている場合）
- すべてが有効であれば、資格情報を保存し、今後のログインのために公開鍵を信頼できます。

登録成功

クライアントが送信した応答が有効な場合、**OnWebAuthnRegistrationSuccessful** イベントが呼び出され、資格情報レコードは将来のログイン検証のためにデータベースに安全に保存できます。

```
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

クライアントレスポンスの検証中にエラーが発生した場合、**OnWebAuthnRegistrationError** イベントが呼び出され、aError パラメータでエラーの理由にアクセスできます。

```
void __fastcall OnWebAuthnRegistrationError(TObject *Sender,
  TsgcWebAuthn_RegistrationVerify_Request* aRequest,
  TsgcWebAuthn_Registration* aRegistration,
  const String aError)
{
  Log("#webauthn_registration_error: " + aError);
}
```

WebAuthn | 認証

WebAuthn 認証は、以前に登録された公開鍵クレデンシャルを使用してユーザーがログインできるようにします。登録時のユーザーの保存済み公開鍵を使用して署名済みチャレンジを検証します。

認証オプション

認証には、クライアントが `TsgcWSAPIServer_WebAuthn` サーバーで設定された **Authentication Options Endpoint** への新しい HTTP リクエストを使用して認証フローを開始する必要があります。デフォルトでは、エンドポイントは `sgcWebAuthn/Authentication/Options` です。したがって、サーバーがドメイン `https://www.test.com` でリスンしている場合、クライアントは URL `https://www.test.com/sgcWebAuthn/Authentication/Options` に新しいリクエストを行う必要があります。

クライアントはアサーション（認証レスポンス）を POST でサーバーに送信します

```
{"username": "test", "user_verification": "preferred"}
```

サーバーは `PublicKeyCredentialRequestOptions` を生成します

```
{
  "challenge": "9d0d61edf30b45f8b88aef7087f9117716e2b7d8b0ee4460b06142f39dd0ec9f",
  "timeout": 60000,
  "rpId": "localhost",
  "allowCredentials": [
    {
      "id": "yeA4BVRlrAfLG-KzqsL_rlI4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
      "type": "public-key",
      "transports": [
        "nfc",
        "usb"
      ]
    }
  ],
  "userVerification": "preferred",
  "hints": [],
  "attestation": "none",
  "attestationFormats": [],
  "extensions": {}
}
```

認証の検証

認証には、クライアントが `TsgcWSAPIServer_WebAuthn` サーバーで設定された **認証確認エンドポイント** への新しい HTTP リクエストを使用して認証フローを開始する必要があります。デフォルトでは、エンドポイントは `sgcWebAuthn/Authentication/Verify` です。サーバーがドメイン `https://www.test.com` でリスンしている場合、クライアントは `https://www.test.com/sgcWebAuthn/Authentication/Verify` という URL に新しいリクエストを作成する必要があります。

ブラウザーは、ユーザーに認証器（例えば指紋、YubiKey）の使用を求めます。認証器は、credential IDにリンクされた秘密鍵でチャレンジに署名します。返される認証情報には次が含まれます：

- **rawId**：資格情報 ID（base64url）。
- **response.authenticatorData**：オーセンティケーターからのメタデータ（バイナリ）。
- **response.clientDataJSON**：元のチャレンジ、オリジン、およびタイプが含まれます。
- **response.signature**：authenticatorData + hash(clientDataJSON) に対する署名。

クライアントリクエストの JSON 例を以下に示します：

```
{
  "id": "yeA4BVRlrAfLG-KzqsL_rlI4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
  "rawId": "yeA4BVRlrAfLG-KzqsL_rlI4ffhuKHK8uoEkVoab065UkS82Zqlh9VFQHIYwOuOo",
  "response": {
    "authenticatorData": "SZYN5YgOjGh0NBcPZHZgW4_krrmihjLHmVzzuoMdl2MFAAAABw",
    "clientDataJSON": "eyJ0eXBldjoid2ViYXV0aG4uZ.....",
    "signature": "MEQCIAJRqvvyys8....",
    "userHandle": "36b9d6a84204487382fee62e7e67a80d"
  },
  "type": "public-key",
  "clientExtensionResults": {},
  "authenticatorAttachment": "cross-platform"
}
```

サーバーはクライアントからのリクエストを読み取り、資格情報が保存されていることを検証し、署名を検証します。署名が有効な場合、イベント **OnWebAuthnAuthenticationSuccessful** が呼び出されます。

認証フロー

次のリンクを使用して、認証フロープロセスの詳細情報を取得してください。

- [WebAuthn 認証リクエスト](#)
- [WebAuthn 認証応答](#)
- [WebAuthn 認証結果](#)

WebAuthn 認証 | リクエスト

ユーザーがログインを試みると、ブラウザーはサーバーに認証オプション（「assertion options」とも呼ばれます）を求めるリクエストを送信します。

```
{
  "username": "alice@example.com"
}
```

このリクエストにより、サーバーは以下のことが可能になります。

- チャレンジ（安全なランダム値）を生成します。
- このユーザーに対して、どの資格情報（公開鍵ID）が有効かを検索します。
- ブラウザーが `navigator.credentials.get()` で使用する暗号化パラメータで応答します。

WebAuthn サーバーコンポーネントがこのリクエストを受信すると（通常ブラウザからサーバーエンドポイント / `sgcWebAuthn/Authentication/Options` への HTTPS POST リクエストで実行されます）、**OnWebAuthnAuthenticationOptionsRequest** イベントが呼び出され、ユーザー名に関連する資格情報（存在する場合）を追加できます。

```
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

ブラウザ（クライアント）がユーザー名（または類似の識別子）をサーバーに送信すると、サーバーはクライアントがブラウザの `navigator.credentials.get()` API を通じて認証プロセスを開始するために使用する認証オプションを返します。

このレスポンスは、ユーザーの認証器（セキュリティキー、生体認証デバイスなど）を使用して WebAuthn 認証セッションを生成するためにブラウザが必要とするパラメータと制約を提供します。

JSON レスポンスの例:

```
{
  "challenge": "Z3lVbWV5YXBpbmdvZG90IQ",
  "timeout": 60000,
  "rpId": "example.com",
  "allowCredentials": [
    {
      "type": "public-key",
      "id": "dXNlckNyZWJZA",
      "transports": ["usb", "nfc", "ble"]
    }
  ],
  "userVerification": "preferred"
}
```

フィールドの説明を以下に示します:

- **challenge:** リプレイ攻撃を防ぐための暗号化チャレンジ（ランダムに生成）。リクエストごとに一意である必要があります。
- **timeout:** オプション。ブラウザが操作をキャンセルする前に待機する時間（ミリ秒単位）。
- **rpId:** リライティングパーティ識別子。通常はドメイン名です。
- **allowCredentials:** ユーザーが以前に登録した許容可能な資格情報（資格情報 ID）のリスト。
- **userVerification:** ブラウザにユーザーの確認方法を通知します: "required"、"preferred"、または "discouraged"。
- **extensions (オプション) :** 認証器から要求する追加の機能またはデータ。

レスポンスがクライアントに送信される前に **OnWebAuthnAuthenticationOptionsResponse** イベントが呼び出され、レスポンスをカスタマイズできます。

WebAuthn Authentication | Result

目的は、ブラウザが提供した署名済みアサーションを検証し、ユーザーが最初に登録された秘密鍵を所有していることを証明することです。これがユーザーを安全にログインさせます。ユーザーがオーセンティケーター（例：指紋認証、セキュリティキー）を操作した後、ブラウザは認証結果とともに POST リクエストをサーバーに送り返します。以下に JSON の例を示します：

```
{
  "id": "credential-id",
  "rawId": "base64url-encoded-credential-id",
  "type": "public-key",
  "response": {
    "clientDataJSON": "base64url",
    "authenticatorData": "base64url",
    "signature": "base64url",
    "userHandle": "optional"
  }
}
```

サーバーが構成されたエンドポイント（例: /sgcWebAuthn/Authentication/Verify）でこのリクエストを受信すると、次のステップを検証する必要があります。

- DB から認証情報を取得：認証情報 ID を使用してユーザーの登録済み公開鍵とメタデータ（例：署名カウンター）を検索します。
- clientDataJSON のデコードと解析
- 認証データの検証
- 署名の検証

すべての検証が正しい場合、認証は成功し、**OnWebAuthnAuthenticationSuccessful** イベントが呼び出されます。

いずれかのチェックが失敗した場合、**OnWebAuthnAuthenticationError** イベントがエラーの理由とともに呼び出されます。

WebAuthn | MDS

Metadata Service (MDS)は、FIDO認証プログラムを通じて認定された認証器に関する**Metadata Statements**を集約して公開する、**FIDO Alliance**が提供する**集中型サービス**です。これらのステートメントには、認証器に関する詳細なセキュリティ、コンプライアンス、運用情報が含まれます。

- サービスエンドポイントはしばしば**MDS3** (Metadata Serviceプロトコルの第3バージョン) と呼ばれます。
- 信頼者 (WebAuthn を実装する Web サイトやアプリケーションなど) は、認証器に関する情報に基づいた信頼の決定を行うために、MDS から**メタデータステートメント**を取得します。

MDS は WebAuthn を使用するリライティングパーティに対して重要な信頼とセキュリティ検証のレイヤーを追加します。

- **オーセンティケーターの検証:** FIDOスタンダードへのオーセンティケーターの準拠を検証し、WebAuthn証明書に提示されたAAGUIDの検証を支援します。
- **侵害とリボケーション検出:** 侵害または失効した認証器に関する最新情報を提供し、依拠当事者が安全でないデバイスをブロックできるようにします。
- **セキュリティ保証:** 特定の FIDO 認証レベルやユーザー検証強度を満たす認証器のみを許可するなど、セキュリティポリシーの適用に役立ちます。
- **相互運用性:** 異なるオーセンティケーターを使用する異なるブラウザ、プラットフォーム、デバイス間で一貫した動作とセキュリティの期待値を確保します。

設定

プロパティ `WebAuthnOptions.MDS` を使用して MDS の使用を設定できます。主要なプロパティを以下に示します。

- **Enabled:** True (デフォルト値) の場合、WebAuthn リクエストは設定された MDS ファイルに対して検証されます。
- **MDS_FileName:** MDSファイルが保存されているパス。次のURLからダウンロードできます: <https://mds3.fidoalliance.org/>
- **RootCert_FileName:** Root Certificateが保存されているパス。証明書チェーンを検証するために定義する必要があります。 <https://valid.r3.roots.globalsign.com/> からダウンロードできます
- **Leaf_CertificateCRL:** true の場合 (デフォルトは false)、blob のリーフ証明書が CRL (証明書失効リスト) に対して検証されます。
- **CRL_FileName:** CRL ファイルが保存されているパス。定義されておらず Leaf_CertificateCRL が有効になっている場合は、CRL を自動的にダウンロードしようとします。

WebAuthn 認証

認証に成功した後、新しいWebSocketまたはHTTP接続を開くために使用できるBearerトークンをサーバーに送信させたい場合は、パラメータtoken = trueを渡します。例:

```
{
  "username": "alice@example.com", "token": true
}
```

認証が成功すると、サーバーは次のようなレスポンスを送信します:

```
{
  "verified": "ok",
  "authentication": {
    "token": "C760C1C39E3D4E829693A13F18F5CFDE537B516336FC48F7BAB0276176F9E6DE"
  }
}
```

イベント**OnWebAuthnUnauthorized** は、リクエストが認可されておらず、接続が切断される場合に呼び出されます。ここで、どのエンドポイントがWebAuthn認証を必要とし、どのエンドポイントが必要としないかを構成できます。

WebAuthn 認可 | HTTP

新しいトークンを取得したら、このベアータークンを含む認可ヘッダーを送信するだけです。TsgcHTTP1Client の CustomHeaders プロパティを使用して Bearer Token を設定できます。例:

```
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 認証 | WebSocket

新しいトークンを取得したら、このベアータークンを含む認可ヘッダーを送信するだけです。OnHandShake イベントを使用して接続リクエストに Bearer Token を追加できます。例:

```
void __fastcall OnClientHandshake(TsgcWSConnection *Connection, TStringList *Headers)
{
    Headers->Add("Authorization: Bearer C760C1C39E3D4E829693A13F18F5CFDE537B516336FC48F7BAB0276176F9E6DE");
}
```

Webauthn | Javascript クライアント

WebAuthn (Web Authentication API) は、**公開鍵暗号**を使用してセキュアなパスワードレス認証を可能にする W3C 標準です。パスワードの代わりに、ユーザーは**ハードウェアベースの認証器**（指紋リーダー、Face ID、YubiKey など）またはプラットフォーム認証器（Touch ID など組み込み型）を使用して登録・認証します。

Javascript クライアントを使用した登録と認証の処理方法を以下に示します。

WebAuthn 登録

WebAuthn 登録の仕組み

- **ユーザーが登録を開始する:**
 - ユーザーはユーザー名を入力し、**登録**をクリックします。
- **ブラウザがサーバーからオプションをリクエストする:**
 - フロントエンドが登録オプションを取得するために POST リクエストを行います。
- **ブラウザが認証情報を作成:**
 - `SimpleWebAuthnBrowser.startRegistration()` を通じた `navigator.credentials.create()` API を使用して、クレデンシャルが作成されます。
- **サーバーが登録を検証します:**
 - ブラウザは資格情報をサーバーに送り返します。
 - サーバーは登録を検証し、そのユーザーの公開鍵を保存します。

TsgcWSAPIServer_WebAuthn コンポーネントには WebAuthn プロトコルをテストするための HTML ファイルが含まれています。この HTML ファイルには、WebAuthn と対話するための最小限の UI と JavaScript が含まれています。

sgcWebAuthn.html のウォークスルー

Structure Overview:

- **Username Input:** ユーザーの識別子をキャプチャします。
- **ボタン:**
 - Register – WebAuthn 登録を開始します。
 - Authenticate – ログインを開始します（同様に処理されます）。
- **Debug Console:** リアルタイムのデバッグ情報（WebAuthn からの JSON）を表示します。

1. 入力用 HTML UI

```
<input type="text" id="username" name="username" autocomplete="username webauthn" />
<button id="btnRegBegin"><strong>Register</strong></button>
```

2. JavaScript: ボタンクリックハンドラ

```
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. サーバー応答（HTML内の疑似エンドポイント）

```
fetch('/sgcWebAuthn/Registration/Options', ...)
fetch('/sgcWebAuthn/Registration/Verify', ...)
```

4. 登録の完了

```
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 認証

WebAuthn認証の仕組み

- ユーザーが認証を開始する:
 - ユーザーはユーザー名を入力し、登録をクリックします。
- ブラウザがサーバーからオプションをリクエストする:
 - フロントエンドは認証オプションを取得するために POST リクエストを送信します。
- ブラウザーが認証情報を作成:
 - `navigator.credentials.get()` API を `SimpleWebAuthnBrowser.startAuthentication()` 経由で使用します。
- サーバーが認証を確認します:
 - ブラウザは資格情報をサーバーに送り返します。
 - サーバーは署名された結果を検証します。

1. HTML UI のセットアップ

```
<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. 認証オプションを取得

`startAuthentication` を呼び出す前に、ユーザー名をサーバーに送信します。

```
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'
  }),
});
```

サーバーは以下を含む JSON オブジェクトで応答します：

```
{
  "challenge": "base64url-encoded-random-string",
  "allowCredentials": [
    {
      "id": "base64url-credential-id",
      "type": "public-key"
    }
  ],
  "userVerification": "preferred",
  "rpId": "yourdomain.com"
}
```

これは `PublicKeyCredentialRequestOptions` と呼ばれます。

3. 認証器のレスポンスを受信

`asseResp` は次のようになります（簡略版）：

```
{
  "id": "credentialId",
  "rawId": "base64url-encoded-id",
  "response": {
    "authenticatorData": "...",
    "clientDataJSON": "...",
    "signature": "...",
    "userHandle": "..."
  },
  "type": "public-key",
  "clientExtensionResults": {}
}
```

この応答は、ユーザーが次のことを証明します：

- 認証子に保存された秘密鍵を所有しています
- サーバーのチャレンジに署名しました。
- （必要に応じて）物理的に存在していた

4. 署名された認証応答をサーバーに送信する

ユーザーが認証器と対話した後 (`startAuthentication()` を介して)、JavaScript でレスポンスオブジェクトを受け取ります。

```
const verificationResp = await fetch('/webauthn/authenticate/verify', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(asseResp), // this is the signed response
});
```

5. サーバーのレスポンスを取得する

サーバーは次のような結果を返します。

```
{ "verified": true }
```

または何か問題が発生した場合：

```
{ "verified": false, "error": "Invalid signature" }
```

そしてフロントエンドコードで処理します：

```
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 SQS とは何ですか？

Amazon Simple Queue Service (SQS) は、マイクロサービス、分散システム、およびサーバーレスアプリケーションの分離とスケーリングを可能にするフルマネージドメッセージキューイングサービスです。SQS はメッセージ指向ミドルウェアの管理と運用に関連する複雑さとオーバーヘッドを排除し、開発者が差別化作業に集中できるようにします。SQS を使用すると、他のサービスの利用可能性を必要とせず、メッセージを失うことなく、あらゆるボリュームでソフトウェアコンポーネント間でメッセージを送信、保存、受信できます。

メリット

管理コストを削減する

SQS には初期費用がなく、メッセージングソフトウェアの取得、インストール、設定が不要で、サポートインフラの時間のかかる構築とメンテナンスも必要ありません。

メッセージを確実に配信する

SQS を使用すると、アプリケーションコンポーネントを切り離して独立して実行および失敗できるようになり、システム全体のフォールトトレランスが向上します。

機密データを安全に保護する

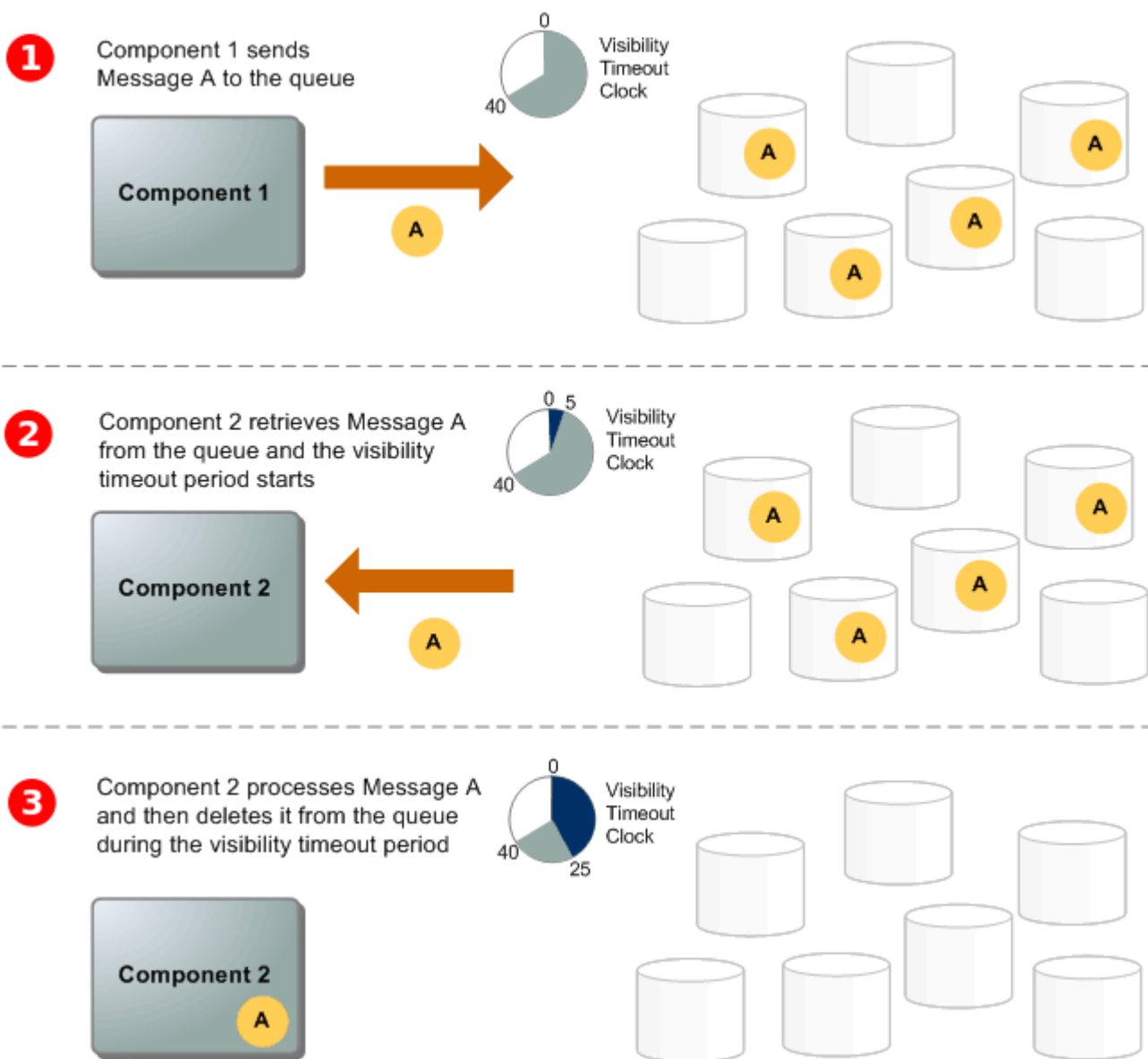
Amazon SQS を使用して、サーバー側暗号化 (SSE) で各メッセージボディを暗号化することで、アプリケーション間で機密データを安全に交換できます。

弾力的かつコスト効率よくスケール

SQS はアプリケーションとともに弾力的にスケールするため、キャパシティプランニングや事前プロビジョニングについて心配する必要はありません。

ワークフロー

以下のシナリオは、Amazon SQS メッセージのキューにおけるライフサイクル（作成から削除まで）を説明しています。



Amazon SQS の使い方

始める前に、Amazon SQS の設定手順を完了してください。

ステップ 1: キューを作成する

1. Amazon SQS コンソールにサインインします。
2. 新しいキューの作成を選択してください。
3. [新しいキューの作成] ページで、正しいリージョンにいることを確認してから、キュー名を入力します。
4. デフォルトでは Standard が選択されています。FIFO を選択してください。
5. デフォルトパラメータでキューを作成するには、Quick-Create Queue を選択します。

新しいキューが作成され、キューリストで選択されます。

ステップ 2: メッセージを送信する

キューを作成した後、メッセージを送信できます。以下の例は既存のキューにメッセージを送信する方法を示しています。

1. キューリストから、作成したキューを選択します。
2. キューアクションから「メッセージを送信」を選択します。
3. メッセージが送信され、「QueueName にメッセージを送信」ダイアログボックスに送信されたメッセージの属性が表示されます。

ステップ 3: メッセージの受信と削除

キューにメッセージを送信した後、それを取得（コンシューム）することができます。キューからメッセージをリクエストする際、取得するメッセージを指定することはできません。代わりに、取得したいメッセージの最大数（最大10件）を指定します。

ステップ 4: キューの削除

Amazon SQS キューを使用しておらず（近い将来も使用する予定がない場合）、Amazon SQS からキューを削除することがベストプラクティスです。

SQS クライアント

```
// TsgcHTTPAWS_SQS_Client is the component used for connect to Amazon SQS.
// Client connects using HTTPs protocol and authenticates using Access Key provided by Amazon.

// Before you attempt to connect to SQS service, you must set some data in AWSOptions property.

// Region: your endpoint region, example: us-east-1.
// AccessKey: access key provided by Amazon.
// SecretKey: secret key provided by Amazon.

// The following methods are supported by SQS client:

// AddPermission
// Adds a permission to a queue for a specific principal. This allows sharing access to the queue.

// ChangeMessageVisibility
// Changes the visibility timeout of a specified message in a queue to a new value. The default visibility
// The minimum is 0 seconds. The maximum is 12 hours.

// ChangeMessageVisibilityBatch
// Changes the visibility timeout of multiple messages. This is a batch version of ChangeMessageVisibility.
// The result of the action on each message is reported individually in the response. You can send up to 10

// CreateQueue
// Creates a new standard or FIFO queue. You can pass one or more attributes in the request.

vURL = SQS->CreateQueue("sqs_queue");
if (vURL != "")
{
    DoLog("#CreateQueue: " + vURL);
}

// DeleteMessage
// Deletes the specified message from the specified queue. To select the message to delete, use the Receipt

if (SQS->DeleteMessage("sqs_queue", "...receipt handle goes here...") == true)
{
    DoLog("#DeleteMessage: ok");
}
else
{
    DoLog("#DeleteMessage: error");
}

// DeleteMessageBatch
// Deletes up to ten messages from the specified queue. This is a batch version of DeleteMessage. The result

// DeleteQueue
// Deletes the queue specified by the queue name, regardless of the queue's contents.

if (SQS->DeleteQueue(txtQueueName->Text) == true)
{
    DoLog("#Delete Queue: ok");
}
else
```

```

{
    DoLog("#Delete Queue: error");
}

// GetQueueAttributes
// Gets attributes for the specified queue.

TsgcSQSAttributes *oAttributes = new TsgcSQSAttributes();
try
{
    if (SQS->GetQueueAttributes("sqs_queue", oAttributes) == true)
    {
        for (int i = 0; i < oAttributes->Count; i++)
        {
            DoLog("#Attribute: " + static_cast<TsgcSQSAttribute*>(oAttributes->Item[i])
                ->AttributeName + " " + static_cast<TsgcSQSAttribute*>(oAttributes->Item[i])
                ->AttributeValue);
        }
    }
    else
    {
        DoLog("#GetQueueAttributes: error");
    }
}
finally
{
    oAttributes->Free();
}

// GetQueueUrl
// Returns the URL of an existing Amazon SQS queue.

// ListDeadLetterSourceQueues
// Returns a list of your queues that have the RedrivePolicy queue attribute configured with a dead-letter

// ListQueueTags
// List all cost allocation tags added to the specified Amazon SQS queue.

// PurgeQueue
// Deletes the messages in a queue specified by the QueueName parameter.

if (SQS->PurgeQueue("sqs_queue") == true)
{
    DoLog("#PurgeQueue: ok");
}
else
{
    DoLog("#PurgeQueue: error");
}

// ReceiveMessage
// Retrieves one or more messages (up to 10), from the specified queue.

TsgcSQSReceiveMessageResponses *oResponses = new TsgcSQSReceiveMessageResponses();
try
{
    if (SQS->ReceiveMessage("sqs_test", oResponses) == true)
    {
        for (int i = 0; i < oResponses->Count; i++)
        {
            DoLog("#ReceiveMessage: " + static_cast<TsgcSQSReceiveMessageResponse*>(oResponses->Item[i])->Body)
                FRceiptHandle = 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 take
// 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.
```

イベント

OnSQSBeforeRequest

このイベントは、sqsコンポーネントがHTTPリクエストを行う前に呼び出されます。URLパラメータにアクセスでき、HandledパラメータがTrueに設定されている場合は、コンポーネントがHTTPリクエストを行わないことを意味します。

OnSQSError

コンポーネントがリクエストを行う際にエラーが発生した場合、このイベントはエラーコードとエラーの説明とともに呼び出されます。

OnSQSResponse

このイベントは、サーバーからの生のレスポンスを含む HTTP リクエストの後に呼び出されます。

Google Cloud | Google OAuth2 キー

sgcWebSockets Google Cloud コンポーネントを使用して OAuth2 で認証するには、まず Google Cloud から OAuth2 キーを取得する必要があります。

Google OAuth2 キーを取得し、PubSubサンプルアプリケーションで設定する手順を以下に示します。

Google Cloud アカウントに最初にログインし、既存のプロジェクトを使用するか新しいプロジェクトを作成します。

その後、**認証情報** メニューに移動し、**認証情報を作成**ボタンをクリックして、オプション **OAuth クライアント ID** を選択します。

アプリケーションタイプを選択し、わかりやすい名前を設定します。

成功すると、クライアント ID と クライアントシークレットが取得されます。

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データを誰とも共有しないでください！

次に、sgcWebSockets PubSubサンプルにコピーし、Project Id(プロジェクト名ではない)を追加します

Select a project



NEW PROJECT

RECENT

ALL

Name		ID
✓  PubSub 		pubsub-270909
 GmailApiTest 		gmailapitest-265010
 My Project 		melodic-voice-265009

CANCEL

OPEN

これは sgcWebSocket PubSub サンプルでの設定方法です。

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

その後、例えば新しいトピックの作成を試みることができます。最初は、OAuth2接続を認可する必要があるため、Googleが提供するOAuth2認証情報でアカウントへのアクセスを認可するために、新しいWebブラウザが表示されます。

 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](#)

接続を許可し、成功した場合はこのAPIを使用して作業を開始できます。

Google Pub/Sub API Client

Google API

Client Id

>4180rlbdf6mc32e3ncmn0.apps.googleusercontent.com

Client Secret

pvogD9reEOt9iLL6eR1jE60Z

Publisher

Project-id

pubsub-270909

Topic

topic-1

Create Topic

Delete Topic

Get Topic

List Topics

Publish

Message sent from sgcWebSockets Pub/Sub API.

Subscriber

Project-id

Subscription

subscription-1

Topic

topic-1

Create Subs.

Delete Subs.

Get Subs.

List Subs.

Pull

Acknowledge

#CreateTopic: { "name": "projects/pubsub-270909/topics/topic-1" }

#Publish: { "messageIds": ["1780785665748120"] }

#Publish: { "messageIds": ["1780799518708503"] }

Google Cloud | Google Service Accounts

sgcWebSockets Google Cloud コンポーネントを使用してサービスアカウントで認証するには、まずプライベートキー証明書を取得する必要があります

Google Cloud から。

Google プライベートキー証明書を取得する手順と、PubSub サンプルアプリケーションでの設定方法を以下に示します。

Google Cloud アカウントに最初にログインし、既存のプロジェクトを使用するか新しいプロジェクトを作成します。

The screenshot shows the 'IAM & Admin' console. The left sidebar has 'Service Accounts' selected. The main area is titled 'Service accounts for project "PubSub"'. It includes a '+ CREATE SERVICE ACCOUNT' button and a 'DELETE' button. Below this, there is explanatory text about service accounts and a table with columns: Email, Status, Name, Description, Key ID, Key creation date, and Actions. The table currently shows 'No rows to display'.

CREATE SERVICE ACCOUNT を選択すると新しいページが表示され、サービスアカウント名と説明を設定する必要があります

The screenshot shows the 'Create service account' page. The left sidebar has 'Service Accounts' selected. The main area is titled 'Create service account' and has a step indicator '1 Service account details'. It contains three input fields: 'Service account name' (filled with 'sgcServiceAccount'), 'Service account ID' (filled with 'sgcserviceaccount' and '@pubsub-270909.iam.gserviceaccount.com'), and 'Service account description' (filled with 'Service Account Test'). A 'CREATE' button is at the bottom.

次に、少なくとも 1 つのロールを選択します。ここでは、クライアントがトピックをパブリッシュおよびサブスクライブできるように PubSub Admin を選択しますが、より少ない権限を持つ別のロールを選択することもできます。

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](#)

続行を押して、最終的に他のユーザーへのアクセスを許可できます

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](#)

完了したら DONE を押すと、新しいレコードが表示されます。

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

次のステップは新しいキーを作成することです。アクション列の「キーを作成」オプションを選択し、JSON 形式で設定をダウンロードするために JSON を選択してください。新しいキーが作成されます。

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	⋮

最後に、sgcWebSockets PubSub クライアントに Google が提供するデータを入力するだけです。設定の JSON ファイルを読み込むには **LoadSettingsFromFile** を使用できます。

Google Pub/Sub API Client

OAuth2 Service Account

Client Email: sgcserviceaccount@pubsub-270909.iam.gserviceaccount.com

Private Key Id: 425a876f68b8b2a66f2fcc5b2dee8e918b0eb9ab

Private Key: -----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBCwggSjAgEAAoIBA
QC3xz0L2Wkr9Hvj
HOMMf5Qum78NI1c8nVcF8HlQVc8amSLV

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/Sub とは何ですか？

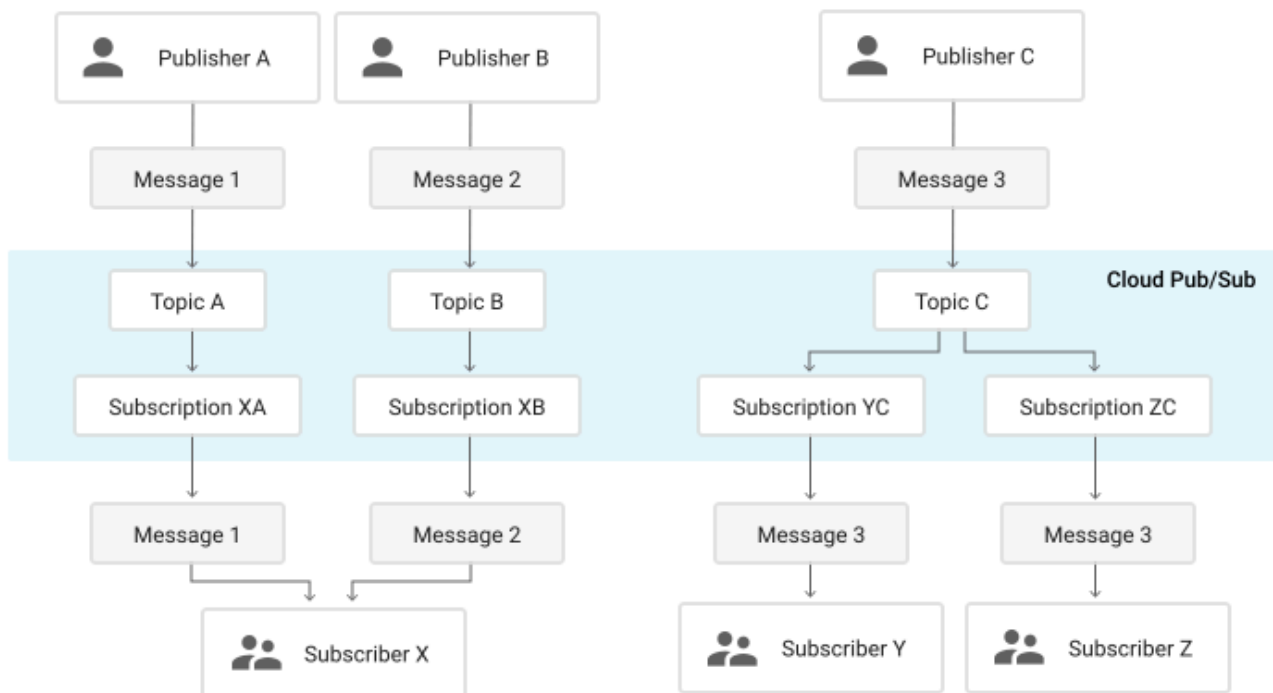
Pub/Subは、エンタープライズのメッセージ指向ミドルウェアの柔軟性と信頼性をクラウドにもたらしめます。同時に、Pub/Subはスケーラブルで耐久性のあるイベント取り込みおよび配信システムであり、最新のストリーム分析パイプラインの基盤として機能します。送信者と受信者を分離する多対多の非同期メッセージングを提供することで、独立して記述されたアプリケーション間の安全かつ高可用性の通信を可能にします。Pub/Subは低レイテンシで耐久性のあるメッセージングを提供し、開発者がGoogle Cloud Platform上でホストされたシステムや外部のシステムを迅速に統合するのに役立ちます。

機能

少なくとも 1 回の配信 同期的なクロスゾーンメッセージレプリケーションとメッセージごとの受信追跡により、あらゆる規模で少なくとも 1 回の配信が保証されます。	Open 7つの言語でのオープン API とクライアントライブラリがクロスクラウドおよびハイブリッドデプロイメントをサポートしています。	Exactly-once 処理 Cloud Dataflow は Cloud Pub/Sub ストリームの信頼性が高く、表現力豊かな exactly-once 処理をサポートします。
デフォルトでグローバル 世界中どこからでもパブリッシュ・コンシュームでき、一貫したレイテンシーを実現します。レプリケーションは不要です。	プロビジョニング不要、すべて自動 Cloud Pub/Sub にはシャードやパーティションがありません。クォータを設定してパブリッシュとコンシュームをするだけです。	コンプライアンスとセキュリティ Cloud Pub/Sub は HIPAA 準拠のサービスで、細かいアクセス制御とエンドツーエンドの暗号化を提供します。
統合 Cloud Storage や Gmail の更新イベント、サーバーレスのイベント駆動型コンピューティングのための Cloud Functions など、複数のサービスとの統合を活用できます。	シークと再生 バックログの任意の時点またはスナップショットに巻き戻して、メッセージを再処理できます。古いデータを破棄するために早送りします。	

パブリッシャー/サブスクライバーの関係

パブリッシャーアプリケーションはトピックにメッセージを作成して送信します。サブスクライバーアプリケーションはトピックへのサブスクリプションを作成してメッセージを受信します。通信は一对多（ファンアウト）、多対一（ファンイン）、多対多が可能です。



一般的なユースケース

- **ネットワーククラスターでのワークロードのバランシング。** 例えば、大きなタスクキューを Google Compute Engine インスタンスなどの複数のワーカーに効率的に分散できます。
- **非同期ワークフローの実装。** 例えば、注文処理アプリケーションは注文をトピックに発注でき、そこから1人以上のワーカーによって処理できます。
- **イベント通知の配信。** 例えば、ユーザー登録を受け付けるサービスは、新しいユーザーが登録されるたびに通知を送信でき、ダウストリームサービスはイベントの通知を受け取るために購読できます。
- **分散キャッシュの更新。** 例えば、アプリケーションが変更されたオブジェクトのIDを更新するための無効化イベントを公開できます。
- **複数のシステムへのログ記録。** 例えば、Google Compute Engine インスタンスは監視システム、後のクエリのためのデータベースなどにログを書き込むことができます。
- **さまざまなプロセスやデバイスからのデータストリーミング。** 例えば、住宅センサーがクラウドでホストされているバックエンドサーバーにデータをストリーミングできます。
- **信頼性の向上。** 例えば、単一ゾーンのCompute Engineサービスは、共通トピックを購読することで追加ゾーンで動作し、ゾーンまたはリージョンの障害から回復できます。

認証

Google Pub/Sub コンポーネントクライアントは、以下のメソッドを使用して Google サーバーにログインできます:

- **gcaOAuth2:** OAuth2 プロトコル
- **gcaJWT:** JWT トークン。

OAuth2

ログインはウェブブラウザで行われ、ユーザーは自身のアカウントでログインして PubSub リクエストを承認します。

- **GoogleCloudOptions.OAuth2.ClientId**: OAuth2プロトコルを介して認証するためにGoogleが提供する ClientID です。
- **GoogleCloudOptions.OAuth2.ClientSecret**: Google が OAuth2 プロトコルを通じた認証に提供するクライアントシークレット文字列です。
- **GoogleCloudOptions.OAuth2.Scope**: OAuth2 のスコープ。通常、追加のアクセス権限が必要でない限り、デフォルト値を変更する必要はありません。
- **GoogleCloudOptions.OAuth2.LocalIP**: OAuth2 プロトコルは認証サーバーからの応答をリッスンするローカルサーバーを必要とします。これは IP または DNS 名です。デフォルトは 127.0.0.1 です。
- **GoogleCloudOptions.OAuth2.LocalPort**: ローカルサーバーのリスニングポート。
- **GoogleCloudOptions.OAuth2.RedirectURL**: LocalPort + LocalIP とは異なるリダイレクト URL を設定する必要がある場合、このプロパティに設定できます (例: http://127.0.0.1:8080/oauth2)。

サービスアカウント

ログインは、Google が提供する秘密鍵を使用してリクエストに署名することによって行われます。この方法は、ユーザーインタラクションなしの自動化されたサービスやアプリケーションに推奨されます。

- **GoogleCloudOptions.JWT.ClientEmail**: 新しいサービスアカウントを作成するときに提供されるクライアントメール名。JSON 設定ファイルの "client_email" ノード。
- **GoogleCloudOptions.JWT.PrivateKeyId**: Googleによって提供される Private Key ID。JSON設定ファイルの「private_key_id」ノード。
- **GoogleCloudOptions.JWT.PrivateKey**: Googleが提供する秘密鍵証明書です。JSON構成ファイルの「private_key」ノードです。

TLS オプション

TLSOptionsは、Google Cloud Pub/Subとの通信に使用されるセキュアチャンネルの構成を一元管理します。

- **TLSOptions.IOHandler**: ご利用のデプロイ環境に適したTLS実装を選択します (OpenSSL、SChannelなど)。
- **TLSOptions.Version**: インフラストラクチャで必要な場合は、ネゴシエーションを特定の TLS バージョンに制限します。
- **TLSOptions.VerifyCertificate**: より厳格なセキュリティポリシーを有効にする証明書検証を制御します。
- **TLSOptions.OpenSSL_Options.LibPath**: アプリケーションに同梱されている OpenSSL バイナリが含まれるディレクトリを設定します。
- **TLSOptions.SChannel_Options.UseLegacyCredentials**: SChannel が必要とする場合に Windows のレガシー資格情報処理を有効にします。

例:

```
TsgcHTTPGoogleCloud_PubSub_Client *oPubSub = new TsgcHTTPGoogleCloud_PubSub_Client(NULL);
oPubSub->TLSOptions->IOHandler = ioOpenSSL;
oPubSub->TLSOptions->Version = tls1_3;
oPubSub->TLSOptions->VerifyCertificate = true;
oPubSub->TLSOptions->OpenSSL_Options->LibPath = oslpDefaultFolder;
```

新しいサービスアカウントが作成されたとき、すべての設定を含む JSON ファイルをダウンロードできます。このファイルは PubSub コンポーネントで処理できます。メソッド **LoadSettingsFromFile** を呼び出し、JSON ファイル名を引数として渡してください。

最も一般的な使用方法

- 設定
 - **Google OAuth2 キー**
 - **サービスアカウント**

Google Pub/Sub クライアント

OAuth2

Google Pub/Sub API で作業するために、sgcWebSockets Pub/Sub コンポーネントはデフォルト認証として OAuth2 を使用します。まず Google アカウントから **ClientId** と **ClientSecret** を設定する必要があります。

```
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...";
```

サービスアカウント

サービスアカウントは JWT を構築し、認可トークンとして渡す必要があります。

```
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...";
```

これは、すべての REST API 呼び出しに使用される Google から認証トークンキーを取得するために必要です。

すべてのメソッドはレスポンスを返します。それは成功またはエラーを返す場合があります。

Projects.Snapshots

メソッド	パラメータ	説明	使用例
CreateSnapshot	project, snapshot, subscription	リクエストされたサブスクリプションからスナップショットを作成します。スナップショットは subscriptions.seek 操作で使用され、メッセージの確認応答を一括管理できます。つまり、既存のサブスクリプション内のメッセージの確認状態をスナップショットが取得した状態に設定できます。	CreateSnapshot('pubsub-270909', 'snapshot-1', 'subscription-1')
DeleteSnapshot	project, snapshot	既存のスナップショットを削除します。	DeleteSnapshot('pubsub-270909', 'snapshot-1')
ListSnapshots	プロジェクト	既存のスナップショットを一覧表示します	ListSnapshots('pubsub-270909')

Projects.Subscriptions

メソッド	パラメータ	説明	使用例
AcknowledgeSubscription			
CreateSubscription	project, subscription, topic	指定されたトピックへのサブスクリプションを作成します。サブスクリプションが既に存在する場合はALREADY_EXISTSを返します。対応するトピックが存在しない場合はNOT_FOUNDを返します。	CreateSubscription('pubsub-27090', 'subscription-1', 'topic-1')
DeleteSubscription	project, subscription	既存のサブスクリプションを削除します。サブスクリプションに保持されているすべてのメッセージは即座に削除されます。	DeleteSubscription('pubsub-27090', 'subscription-1')
GetSubscription	project, subscription	サブスクリプションの設定詳細を取得します。	GetSubscription('pubsub-27090', 'subscription-1')
ListSubscriptions	プロジェクト	一致するサブスクリプションを一覧表示します。	ListSubscriptions('pubsub-27090', 'subscription-1')
ModifyAckDeadlineSubscription	project, subscription, AckIds	特定のメッセージのack期限を変更します。このメソッドは、サブスクライバーがメッセージを処理するためにより多くの時間が必要であることを示す場合、または処理が中断された場合にメッセージを再配信に利用できるようにする場合に便利です。これは後続のメッセージに使用されるサブスクリプションレベルのackDeadlineSecondsを変更しないことに注意してください。	

ModifyPushConfigSubscription	project、subscription	指定したサブスクリプションのPushConfigを変更します。これは、プッシュサブスクリプションをプル方式に変更したり（空のPushConfigで示されます）、その逆を行ったり、プッシュサブスクリプションのエンドポイントURLやその他の属性を変更したりするために使用できます。PushConfigの変更ににかかわらず、メッセージは呼び出し中も継続的に配信用として蓄積されます。	
プル	project、subscription	サーバーからメッセージを取得します。指定されたサブスクリプションに対して保留中の同時プル要求が多すぎる場合、サーバーはUNAVAILABLEを返すことがあります。	<code>pull('pubsub-270909', 'subscription-1')</code>
Seek	project、subscription、timeUTC、snapshot	既存のサブスクリプションを、要求で指定された時点またはスナップショットのいずれかへシークします。スナップショットはsubscriptions.seek操作で使用され、メッセージの確認応答を一括管理できます。つまり、既存のサブスクリプションのメッセージの確認応答状態を、スナップショットでキャプチャされた状態に設定できます。サブスクリプションとスナップショットの両方が同じトピック上にある必要があることに注意してください。	

Projects.Topics

メソッド	パラメータ	説明	使用例
CreateTopic	project、topic	指定された名前 で指定されたト ピックを作成し ます。	CreateTopic('pubsub-270909', 'topic-1')
DeleteTopic	project、topic	指定された名前 のトピックを削 除します。ト ピックが存在し ない場合は NOT_FOUNDを 返します。ト ピックが削除さ れた後、同じ名 前で新しいト ピックを作成で きます。これは 古い設定やサブ スクリプション を持たない全く 新しいトピック です。	DeleteTopic('pubsub-270909', 'topic-1')
GetTopic	project、topic	トピックの設定 を取得します。	GetTopic('pubsub-270909', 'topic-1')
ListTopics	プロジェク ト	一致するトピッ クを一覧表示し ます。	ListTopics('pubsub-270909')
Publish	project、topic、message	1 件以上のメッ セージをトピッ クに追加しま す。トピックが 存在しない場合 は NOT_FOUND を返します。	Publish('pubsub-270909', 'topic-1', 'My First PubSub Message.')

Projects.Topics.Subscriptions

メソッド	パラメータ	説明	使用例
ListTopicSubscriptions	project、topic	このト ピックのサブ スクリプ	ListTopicSubscriptions('pubsub-270909', 'topic-1')

シ
ョ
ン
名
を
一
覧
表
示
し
ま
す。

最も一般的なメソッド

Google Cloud Pub/Sub APIで最もよく使用されるメソッドを以下に示します。

新しいトピックの作成方法

ID が pubsub-270909 のプロジェクトにトピック名 topic-1 の新しいトピックを作成します。

```
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");
```

サーバーからのレスポンス

```
{
  "name": "projects/pubsub-270909/topics/topic-1"
}
```

メッセージを発行する

新しく作成されたトピックに新しいメッセージをパブリッシュします。

```
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.");
```

サーバーからのレスポンス

```
{
  "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();
}
```

新しいサブスクリプションの作成方法

ID pubsub-270909 のプロジェクトに、サブスクリプション名 subscription-1 とトピック topic-1 を持つ新しいサブスクリプションを作成します

```
oPubSub = new TsgcHTTPGoogleCloud_PubSub_Client();
oPubSub->GoogleCloudOptions->Auth2->ClientId = "... your google client id...";
oPubSub->GoogleCloudOptions->Auth2->ClientSecret = "... your google client secret...";
oPubSub->CreateSubscription("pubsub-270909", "subscription-1", "topic-1");
```

サーバーからのレスポンス

```
{
  "name": "projects/pubsub-270909/subscriptions/subscription-1",
  "topic": "projects/pubsub-270909/topics/topic-1",
  "pushConfig": {},
  "ackDeadlineSeconds": 10,
  "messageRetentionDuration": "604800s",
  "expirationPolicy": {
    "ttl": "2678400s"
  }
}
```

サブスクリプションからメッセージを読み取る方法

以前に作成したサブスクリプションからメッセージを読み取ります。

```
oPubSub = new TsgcHTTPGoogleCloud_PubSub_Client();
oPubSub->GoogleCloudOptions->Auth2->ClientId = "... your google client id...";
oPubSub->GoogleCloudOptions->Auth2->ClientSecret = "... your google client secret...";
oPubSub->pubsub->Pull("pubsub-270909", "subscription-1");
```

サーバーからのレスポンス

```
{
  "receivedMessages": [
    {
      "ackId": "PjA-RVNEUAYWLF1GSFE3GQhoUQ5PXiM_NSAorREFC08CKF15MEorQVh0Dj4N",
      "message": {
        "data": "TXkgRmlyc3QgTWVzc2FnZSBmcm9tIHNNY1dlY1NvY2tldHMu",
        "messageId": "1050732082561505",
        "publishTime": "2020-03-14T15:25:31.505Z"
      }
    }
  ]
}
```

メッセージはBase64でエンコードされて受信されるため、内容を読み取るには最初にデコードする必要があります。

```
sgcBase_Helpers->DecodeBase64("TXkgRmlyc3QgTWVzc2FnZSBmcm9tIHNNY1dlY1NvY2tldHMu");
```

Google Cloud | カレンダー

Google Calendar API を使用すると、アプリを Google Calendar と統合し、ユーザーとのエンゲージメントを向上させる新しい方法を生み出せます。Calendar API を使用すると、カレンダーイベントの表示、作成、変更だけでなく、カレンダーやアクセス制御など、その他多くのカレンダー関連オブジェクトを操作できます。

API リソース

Google Calendar は以下のリソースを使用します：

- **イベント**： カレンダー上のイベントは、タイトル、開始時刻と終了時刻、出席者などの情報を含みます。イベントは単発または繰り返しにすることができます。イベントは Event リソースで表されます。特定のカレンダーの Events コレクションには、そのカレンダーのすべての Event リソースが含まれています。
- **Calendar**： カレンダーはイベントのコレクションです。各カレンダーには、カレンダーの説明やデフォルトのカレンダータイムゾーンなど、関連するメタデータがあります。単一のカレンダーのメタデータは Calendar リソースで表されます。Calendars コレクションには、既存のすべてのカレンダーの Calendar リソースが含まれます。
- **CalendarList**： Calendar UI のユーザーのカレンダーリスト上のすべてのカレンダーのリスト。カレンダーリストに表示される単一のカレンダーのメタデータは、CalendarListEntry リソースで表されます。このメタデータには、色や新しいイベントの通知など、カレンダーのユーザー固有のプロパティが含まれます。CalendarList コレクションには、特定のユーザーのすべての CalendarListEntry リソースが含まれます。Calendars と CalendarList コレクションの違いの詳細については、Calendar and Calendar List を参照してください。
- **Setting**： カレンダーUIからのユーザー設定（ユーザーのタイムゾーンなど）。単一のユーザー設定は Setting リソースで表されます。Settings コレクションには特定のユーザーのすべての Setting リソースが含まれます。
- **ACL**： ユーザー（またはユーザーグループ）にカレンダーへの特定のアクセスレベルを付与するアクセス制御ルール。単一のアクセス制御ルールは ACL リソースで表されます。特定のカレンダーの ACL コレクションには、そのカレンダーへのアクセスを許可するすべての ACL リソースが含まれます。
- **Color**： Calendar UI に表示される色。Colors リソースは、Calendar UI で利用可能なすべての色のセットを表します。イベント用の色とカレンダー用の色の 2 グループに分かれています。
- **空き/予約済み**： カレンダーにイベントが予定されている時間は「予約済み」とみなされ、イベントがない時間は「空き」とみなされます。Freebusy リソースでは、指定されたカレンダーまたはカレンダーセットの予約済み時間を照会できます。

主な機能

- フル機能の Google Calendar クライアント API V3。
- API でサポートされているすべてのメソッドは、クライアント API を使用して呼び出せます。
- HTTP/2 プロトコルを使用したクライアントリクエスト（*Enterprise エディションのみ）。
- PageNextToken を使用した部分レスポンスの自動処理。
- カレンダーとイベントデータのプロパティへの簡単なアクセス。
- 認証方法：
 - OAuth2: ユーザー操作が必要です。
 - Service Accounts（Domain-Wide Delegation が必要）： Windows サービス、デーモンなど向け。

設定

Google Calendar コンポーネントクライアントには次のプロパティがあります。

OAuth2

- **GoogleCloudOptions.OAuth2.ClientId**： OAuth2 プロトコルを介して認証するために Google が提供する ClientID です。
- **GoogleCloudOptions.OAuth2.ClientSecret**： Google が OAuth2 プロトコルを通じた認証に提供するクライアントシークレット文字列です。

- **GoogleCloudOptions.OAuth2.Scope**: OAuth2 のスコープ。通常、追加のアクセス権限が必要でない限り、デフォルト値を変更する必要はありません。
- **GoogleCloudOptions.OAuth2.LocalIP**: OAuth2 プロトコルは認証サーバーからの応答をリスンするローカルサーバーを必要とします。これは IP または DNS 名です。デフォルトは 127.0.0.1 です。
- **GoogleCloudOptions.OAuth2.LocalPort**: ローカルサーバーのリスニングポート。
- **GoogleCloudOptions.OAuth2.RedirectURL**: LocalPort + LocalIP とは異なるリダイレクト URL を設定する必要がある場合、このプロパティに設定できます (例: http://127.0.0.1:8080/oauth2)。

Scopes プロパティを使用してクライアント API のスコープを変更できます。クライアントでサポートされているスコープを選択してください。

JWT

ログインは、Google が提供する秘密鍵を使用してリクエストに署名することによって行われます。この方法は、ユーザーの操作なしに自動化されたサービスやアプリケーションに推奨されます。[ドメイン全体の委任](#)でサービスアカウントを設定する必要があります。

- **GoogleCloudOptions.JWT.ClientEmail**: 新しいサービスアカウントを作成するときに提供されるクライアントメール名。JSON 設定ファイルの "client_email" ノード。
- **GoogleCloudOptions.JWT.PrivateKeyId**: Googleによって提供されるPrivate Key ID。JSON設定ファイルの「private_key_id」ノード。
- **GoogleCloudOptions.JWT.PrivateKey**: Googleが提供する秘密鍵証明書です。JSON構成ファイルの「private_key」ノードです。
- **GoogleCloudOptions.JWT.Subject**: ドメイン全体の委任を使用してサービスアカウントにリンクされたワークスペースのメールアカウントです。

TLS オプション

Google サーバーとの安全な接続を設定するには、**TLSOptions** プロパティを使用します。

- **TLSOptions.IOHandler**: TLSスタックを選択します (OpenSSL、SChannelなど)。
- **TLSOptions.Version**: 特定の TLS プロトコルバージョンを強制します (例: tls1_2 または tls1_3)。
- **TLSOptions.VerifyCertificate**: サーバー証明書の検証を有効または無効にします。
- **TLSOptions.OpenSSL_Options.LibPath**: アプリケーションと共に展開された OpenSSL ライブラリが含まれるフォルダーを指します。
- **TLSOptions.SChannel_Options.UseLegacyCredentials**: SChannel 使用時に一部の Windows バージョンに必要なレガシー認証情報フローを有効にします。

以下のスニペットは TLS オプションの設定方法を示しています。

```
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;
```

最も一般的な使用方法

- 設定
 - [Google Calendar 更新トークン](#)
 - [Google Calendar サービスアカウント](#)
- 同期
 - [Googleカレンダーのカレンダー同期](#)
 - [Googleカレンダーのイベント同期](#)

カレンダーの同期

TsgcHTTPGoogleCloud_Calendar_Clientコンポーネントを使用すると、直接的なGoogle API呼び出しを使用するか、当社の簡単なCalendarsメソッドを使用してカレンダーを同期できます。

メソッド	パラメータ	説明
NewCalendar	aSummary: カレンダーのタイトル。	新しいカレンダーを作成します
DeleteCalendar	ald: カレンダーの識別子。	既存のカレンダーを削除します。
UpdateCalendar	aResource: カレンダーデータを持つオブジェクト。	既存のカレンダーを更新します。
LoadCalendars		すべてのカレンダーを読み込み、Calendars プロパティにデータを格納します。
LoadCalendarsChanged	aSyncToken: カレンダーの更新に使用された最後のトークン。	Token セットからカレンダーのすべての変更を読み込みます。

カレンダークライアントには **Calendars** というプロパティがあり、前述のメソッドのいずれかを呼び出した後にカレンダーデータにアクセスできます。このプロパティは自動的に同期されます。

イベントの同期

TsgcHTTPGoogleCloud_Calendar_Client コンポーネントを使用すると、直接の Google API 呼び出しや、イベントを同期するための使いやすいイベントメソッドを使用してイベントを同期できます。

メソッド	パラメータ	説明
NewEvent	aCalendarId: カレンダーの ID。 aResource: イベントデータを含むオブジェクト。	新しいイベントを作成します。
DeleteEvent	aCalendarId: カレンダーの ID。 ald: イベントの識別子。	既存のイベントを削除します。
UpdateEvent	aCalendarId: カレンダーの ID。 aResource: イベントデータを含むオブジェクト。	既存のイベントを更新します。
LoadEvents	aCalendarId: カレンダーの ID。	カレンダーのすべてのイベントを読み込みます。
LoadEventsChanged	aCalendarId: カレンダーの ID。 aSyncToken: カレンダーの更新に使用した最後のトークン。	Token セットからカレンダーのすべてのイベントを読み込みます。

Calendars プロパティを使用してイベントデータにアクセスできます。リストから既存のカレンダーを選択し、その **Events** プロパティにアクセスしてください。

Google Calendar API 呼び出し

メソッド	説明
ACL_Delete	アクセス制御ルールを削除します。
ACL_Get	アクセス制御ルールを返します。
ACL_Insert	アクセス制御ルールを作成します。
ACL_List	カレンダーのアクセス制御リストのルールを返します。
ACL_Patch	アクセス制御ルールを更新します。このメソッドはパッチセマンティクスをサポートします。
ACL_Update	アクセス制御ルールを更新します。
ACL_Watch	ACLリソースへの変更を監視します。

メソッド	説明
CalendarList_Delete	ユーザーのカレンダーリストからカレンダーを削除します。
CalendarList_Get	ユーザーのカレンダーリストからカレンダーを返します。
CalendarList_Insert	既存のカレンダーをユーザーのカレンダーリストに挿入します。
CalendarList_List	ユーザーのカレンダーリストにあるカレンダーを返します。
CalendarList_Patch	ユーザーのカレンダーリストで既存のカレンダーを更新します。このメソッドはパッチセマンティクスをサポートします。
CalendarList_Update	ユーザーのカレンダーリストにある既存のカレンダーを更新します。
CalendarList_Watch	CalendarList リソースへの変更を監視します。

メソッド	説明
Calendar_Clear	プライマリカレンダーをクリアします。この操作はアカウントのプライマリカレンダーに関連するすべてのイベントを削除します。
Calendar_Delete	セカンダリカレンダーを削除します。プライマリカレンダーのすべてのイベントをクリアするには <code>calendars.clear</code> を使用してください。
Calendar_Get	カレンダーのメタデータを返します。
Calendar_Insert	セカンダリカレンダーを作成します。
Calendar_Patch	カレンダーのメタデータを更新します。このメソッドはパッチセマンティクスをサポートします。
Calendar_Update	カレンダーのメタデータを更新します。

メソッド	説明
Channel_Stop	このチャンネルを通じてリソースの監視を停止します。

メソッド	説明
Color_Get	カレンダーとイベントのカラー定義を返します。

メソッド	説明
Event_Delete	イベントを削除します。
Event_Get	イベントを返します。
Event_Import	イベントをインポートします。この操作は既存のイベントのプライベートコピーをカレンダーに追加するために使用されます。
Event_Insert	イベントを作成します。

Event_Instances	指定した定期的なイベントのインスタンスを返します。
Event_List	指定されたカレンダーのイベントを返します。
Event_Move	イベントを別のカレンダーに移動します。つまり、イベントのオーガナイザーを変更します。
Event_Patch	イベントを更新します。このメソッドはパッチセマンティクスをサポートします。指定したフィールド値が既存の値を置き換えます。リクエストで指定しないフィールドは変更されません。配列フィールドは、指定された場合、既存の配列を上書きします。これにより、以前の配列要素は破棄されます。
Event_QuickAdd	シンプルなテキスト文字列に基づいてイベントを作成します。
Event_Update	イベントを更新します。
Event_Watch	Events リソースの変更を監視します。

メソッド	説明
Freebusy_Query	カレンダーセットの空き/予約済み情報を返します。

メソッド	説明
Settings_Get	単一のユーザー設定を返します。
Settings_List	認証済みユーザーのすべてのユーザー設定を返します。
Settings_Watch	Settings リソースへの変更を監視します。

アカウント切り替え

アカウントを切り替える場合は、まず **Clear** メソッドを呼び出して現在のカレンダーセッションを消去し、新しいセッションを設定してください。

Google Calendar | カレンダーの読み込み

アカウントからすべてのカレンダーを取得するプロセスは非常に簡単です。次の手順に従ってください。

1. メソッド **LoadCalendars** を呼び出します。
2. メソッドがTrueを返す場合、**Calendars** プロパティにアクセスして、すべてのカレンダーへのアクセスを取得するためにリストを反復できます。

```
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 | イベントの同期

カレンダーからすべてのイベントを取得するプロセスは非常に簡単です。次の手順に従ってください。

1. **LoadEvents**メソッドを呼び出し、**CalendarId** をパラメータとして渡します。
2. メソッドが **True** を返した場合、**Calendars.Events** プロパティにアクセスしてリストを反復処理し、カレンダーのすべてのイベントにアクセスできます。

```
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 API は Google サーバーへの認証に OAuth2 を使用します。sgcWebSockets には認証プロセス全体を処理するコンポーネントがありますが、アプリケーションを閉じて再接続しようとする場合は、2つのオプションがあります。

1. Google API を使用して再認証します
2. リフレッシュトークン（まだ有効な場合）を使用して、認証プロセスを回避します。

RefreshTokenを使用する

初めて認証する際は、OnAuthToken イベントを使用して **RefreshToken** が存在する場合は保存します。例えば INI ファイルに保存できます:

```
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();
    }
}
```

次にアプリケーションを再起動したとき、RefreshToken がある場合は RefreshToken メソッドを呼び出してトークンを引数として渡します(まず Google Calendar API キーを設定する必要があります)。成功すると、再認証なしに Google サーバーにログインできます。

```
GoogleCalendar->RefreshToken("your refresh token here");
```

Google Calendar | サービスアカウント

Google Calendar クライアントはユーザーインタラクションなしにサービスとして動作できるため、Windows サービス、デーモンなどを実行したい場合に役立ちます。

Google Cloud では、このタイプのプロジェクトを実行するために OAuth2 資格情報の代わりに **サービスアカウント** を作成する必要があります。また、Google Calendar API では、カレンダーへのアクセスに必要な資格情報を取得するために、サービスアカウントが **ドメイン全体の委任** を使用する必要があります。

[Google サービスアカウント](#) の作成方法の詳細を読むことができます。

Google Cloud アカウントがサービスアカウントで構成され、Domain-Wide Delegation を使用してワークスペースのメールアカウントにリンクされたら、次の手順に従って Google Calendar クライアントを構成して動作させることができます。

- **GoogleCloudOptions.Authentication** プロパティに **gcaJWT** の値を設定します。
- **LoadSettingsFromFile** メソッドを使用して Google Cloud アカウントで生成された JSON ファイルをインポートします。このファイルには JWT を暗号化するための秘密鍵とクライアントが必要とするその他のプロパティが含まれています。
- JSON ファイルをインポートすると、以下のプロパティが自動的に設定されます:
 - **ClientEmail**: サービスアカウント名です
 - **PrivateKeyId**: 秘密鍵ファイルの id です
 - **PrivateKey**: 秘密鍵ファイルです
- 最後に、プロパティ **GoogleCloudOptions.JWT.Subject** にサービスアカウントに紐付けられた Workspace メールアカウントを設定します。

クライアントを設定したら、ユーザーの操作なしに Google Calendar API へのリクエストを開始できます。

Google Cloud FCM

Firebase Cloud Messaging (FCM) は、無料で確実にメッセージを送信できるクロスプラットフォームのメッセージングソリューションです。

FCMを使用すると、新しいメールや同期可能なその他のデータが利用可能であることをクライアントアプリに通知できます。ユーザーの再エンゲージメントと維持を促進するために通知メッセージを送信できます。インスタントメッセージングなどのユースケースでは、メッセージは最大4096バイトのペイロードをクライアントアプリに転送できます。

コンポーネントはHTTP v1 APIをサポートしています。

認証

Google FCM コンポーネントクライアントは以下のメソッドを使用して Google サーバーにログインできます：

- **gcaOAuth2:** OAuth2 プロトコル
- **gcaJWT:** JWT トークン。

OAuth2

ログインは、ユーザーが自分のアカウントでログインしてFCMリクエストを認可するWebブラウザを使用して行われます。

- **GoogleCloudOptions.OAuth2.ClientId:** OAuth2プロトコルを介して認証するためにGoogleが提供するClientIDです。
- **GoogleCloudOptions.OAuth2.ClientSecret:** Google が OAuth2 プロトコルを通じた認証に提供するクライアントシークレット文字列です。
- **GoogleCloudOptions.OAuth2.Scope:** OAuth2 のスコープ。通常、追加のアクセス権限が必要でない限り、デフォルト値を変更する必要はありません。
- **GoogleCloudOptions.OAuth2.LocalIP:** OAuth2 プロトコルは認証サーバーからの応答をリスンするローカルサーバーを必要とします。これは IP または DNS 名です。デフォルトは 127.0.0.1 です。
- **GoogleCloudOptions.OAuth2.LocalPort:** ローカルサーバーのリスニングポート。
- **GoogleCloudOptions.OAuth2.RedirectURL:** LocalPort + LocalIP とは異なるリダイレクト URL を設定する必要がある場合、このプロパティに設定できます（例: `http://127.0.0.1:8080/oauth2`）。

サービスアカウント

ログインは、Google が提供する秘密鍵を使用してリクエストに署名することによって行われます。この方法は、ユーザーインタラクションなしの自動化されたサービスやアプリケーションに推奨されます。

- **GoogleCloudOptions.JWT.ClientEmail:** 新しいサービスアカウントを作成するときに提供されるクライアントメール名。JSON 設定ファイルの "client_email" ノード。
- **GoogleCloudOptions.JWT.Subject:** 新しいサービスアカウントを作成するときに提供されるクライアントのメール名。JSON 設定ファイルの "client_email" ノード。
- **GoogleCloudOptions.JWT.PrivateKeyId:** Googleによって提供されるPrivate Key ID。JSON設定ファイルの「private_key_id」ノード。
- **GoogleCloudOptions.JWT.PrivateKey:** Googleが提供する秘密鍵証明書です。JSON構成ファイルの「private_key」ノードです。

TLS オプション

Firebase Cloud Messagingの接続は、**TLSOptions**プロパティを通じて調整できます。

- **TLSOptions.IOHandler:** 使用するTLSスタックを選択します（OpenSSL、SChannelなど）。
- **TLSOptions.Version:** プロジェクトに必要な特定の TLS プロトコルバージョンをリクエストします。
- **TLSOptions.VerifyCertificate:** Google への接続時にサーバー証明書の検証を強制します。
- **TLSOptions.OpenSSL_Options.LibPath:** OpenSSL ランタイム ライブラリが配置されている場所を定義します。
- **TLSOptions.SChannel_Options.UseLegacyCredentials:** Windows SChannelがレガシー認証情報のネゴシエーションを必要とする場合に有効にします。

設定例:

```
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;
```

新しいサービスアカウントが作成されると、すべての構成を含むJSONファイルをダウンロードできます。このファイルはFCMコンポーネントで処理できます。**LoadSettingsFromFile**メソッドを呼び出し、JSONファイル名を引数として渡すだけです。

最も一般的な使用方法

- 設定
 - [Google OAuth2 キー](#)
 - [サービスアカウント](#)

Google FCM Client

OAuth2

Google FCM API と連携するために、sgcWebSockets FCM コンポーネントはデフォルトの認証方法として OAuth2 を使用します。そのため、まず Google アカウントから **ClientId** と **ClientSecret** を設定する必要があります。

```
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...";
```

サービスアカウント

サービスアカウントは JWT を構築し、認可トークンとして渡す必要があります。

```
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...";
```

これは、すべての REST API 呼び出しに使用される Google から認証トークンキーを取得するために必要です。

すべてのメソッドはレスポンスを返します。それは成功またはエラーを返す場合があります。

Send Message

通知メッセージを送信するには **SendMessage** メソッドを使用します。この関数には2つの引数があります:

1. Project: Google Cloud アカウントのプロジェクト ID です。
2. Payload: はJSONテキストメッセージです。例:

```
{
  "message": {
    "topic": "news",
    "notification": {
      "title": "Breaking News",
      "body": "New news story available."
    },
    "data": {
      "story_id": "story_12345"
    }
  }
}
```

TsgcWebView2

TsgcWebView2 は Microsoft Edge WebView2 (Chromium) をラップするビジュアル VCL コンポーネントです。フォームにドロップするだけで、Delphi アプリケーションに最新の本格的な Web ブラウザーを組み込みます。Delphi 7 から Delphi 13 までサポートしています。

要件

- Microsoft Edge WebView2 ランタイム (Windows 10/11 に含まれているか、Microsoft からダウンロード可能)
- WebView2Loader.dll (付属、実行ファイルの隣に配置してください)
- Windowsのみ

クイックスタート

フォームに TsgcWebView2 をドロップして URL に移動します:

```
sgcWebView21 := TsgcWebView2.Create(Self);
sgcWebView21.Parent := Self;
sgcWebView21.Align := alClient;
sgcWebView21.DefaultURL := 'https://www.example.com';
```

機能

- ナビゲーション
 - [Navigate](#)、[NavigateToString](#)、[GoBack](#)、[GoForward](#)、[Reload](#)、[Stop](#)、[POST ナビゲーション](#)
- JavaScript
 - [ExecuteScript \(async\)](#)、[ExecuteScriptSync](#)、[AddInitScript](#)
- Cookie Management
 - [GetCookies](#)、[AddOrUpdateCookie](#)、[DeleteCookie](#)、[DeleteAllCookies](#)
- ダウンロードコントロール
 - [OnDownloadStarting](#)、[OnDownloadProgress](#)、[OnDownloadCompleted](#)
- 設定と構成
 - [ScriptEnabled](#)、[DevToolsEnabled](#)、[ContextMenuEnabled](#) など
- 高度な機能
 - [印刷](#)、[オーディオ/ミュート](#)、[証明書](#)、[ファビコン](#)、[仮想ホスト](#)、[スクリーンショット](#)

プロパティ

- **DefaultURL**: string — 初期化後にナビゲートする URL。
- **AutoInitialize**: Boolean — True の場合、ウィンドウハンドルが割り当てられるときに WebView2 が自動的に作成されます。デフォルト値: True。
- **ZoomFactor**: Double型 — ズームレベル (1.0 = 100%)。
- **UserDataFolder**: 文字列 — クッキー、キャッシュ、および権限ストレージのパス。
- **BrowserExecutableFolder**: string — 固定バージョン WebView2 ランタイム配布へのパス。
- **Settings**: TsgcWebView2Settings — 設計時のブラウザー設定 (ScriptEnabled、DevToolsEnabledなど)。
- **URL**: 文字列 (読み取り専用) — 現在のページの URL。
- **DocumentTitle**: string (読み取り専用) — 現在のページタイトル。
- **CanGoBack**: Boolean (読み取り専用) — 戻るナビゲーションが可能かどうか。
- **CanGoForward**: Boolean (読み取り専用) — 前方ナビゲーションが可能かどうか。
- **Initialized**: Boolean (読み取り専用) — WebView2の準備ができていますかどうかです。
- **IsMuted**: Boolean — オーディオのミュート/ミュート解除。
- **IsDocumentPlayingAudio**: Boolean (読み取り専用) — ページが音声を再生中かどうか。
- **FaviconURL**: string (読み取り専用) — 現在のページのファビコンURI。
- **StatusBarText**: string (読み取り専用) — 現在のステータスバーテキスト。
- **CookieManager**: TsgcWebView2CookieManager (読み取り専用) — Cookie 管理オブジェクト。
- **WebView**: ICoreWebView2 (読み取り専用) — 直接 COM インターフェースへのアクセス。
- **Controller**: ICoreWebView2Controller (読み取り専用) — 直接のCOMインターフェイスアクセス。

- **Environment:** `ICoreWebView2Environment` (読み取り専用) — 直接のCOMインターフェイスアクセス。

メソッド

- **Navigate**(aURL: string) — URL に移動します。
- **NavigateToString**(aHTML: string) — HTMLコンテンツを直接読み込みます。
- **GoBack / GoForward** — 履歴をナビゲートします。
- **再読み込み / 停止** — 再読み込みまたは読み込みを停止します。
- **ExecuteScript**(aScript: string) — JavaScript を非同期で実行します (結果は `OnScriptExecuted` で返されます)。
- **ExecuteScriptSync**(aScript: string): string — JavaScript を同期実行し、JSON 結果を返します。
- **AddInitScript**(aScript: string) — すべてのページロード時に実行される JavaScript を追加します。
- **RemoveInitScript**(aScriptId: string) — 以前に追加した初期化スクリプトを削除します。
- **NavigateWithPostData**(aURI, aMethod, aPostData, aHeaders: string) — カスタムHTTPメソッド、本文、ヘッダーでナビゲートします。
- **PrintToPdf**(aFilePath: string) — ページを PDF として保存します。
- **ShowPrintUI** — ネイティブの印刷ダイアログを表示します。
- **CapturePreviewToFile**(aFilePath: string; aFormat: Integer) — スクリーンショット (0=PNG, 1=JPEG)。
- **OpenDevToolsWindow** — Edge DevTools を開きます。
- **OpenTaskManagerWindow** — Edge タスクマネージャーを開きます。
- **SetVirtualHostNameToFolderMapping**(aHostName, aFolderPath: string; aAccessKind: Integer) — ホスト名をローカルフォルダーにマップします。
- **ClearVirtualHostNameToFolderMapping**(aHostName: string) — マッピングを削除します。
- **PostWebMessageAsString**(aMessage: string) — Web コンテンツにメッセージを送信します。
- **PostWebMessageAsJson**(aJson: string) — JSON を Web コンテンツに送信します。
- **GetProfileName**: string — ブラウザプロファイル名を取得します。
- **ClearBrowsingData**(aKinds: Cardinal) — 特定の閲覧データの種別をクリアします。
- **ClearAllBrowsingData** — すべてのブラウジングデータをクリアします。
- **PostSharedBufferToScript**(aSharedBuffer: IUnknown; aAccess: Integer; aAdditionalDataAsJson: string) — メモリバッファをWebコンテンツと共有します。
- **InitializeWebView / FinalizeWebView** — 手動初期化コントロール。

イベント

- **OnInitialized** — `WebView2` の準備ができたとき発生します。
- **OnNavigationStarting**(aURI; alsUserInitiated, alsRedirected: Boolean; var aCancel: Boolean) — ナビゲーション前。
- **OnNavigationCompleted**(alsSuccess: Boolean; aWebErrorStatus: Integer) — ナビゲーション後。
- **OnSourceChanged**(alsNewDocument: Boolean) — URLが変更されました。
- **OnDocumentTitleChanged**(aTitle: string) — ページタイトルが変更されました。
- **OnWebMessageReceived**(aSource, aWebMessageAsJson, aWebMessageAsString: string) — Webコンテンツからのメッセージ。
- **OnNewWindowRequested**(aURI: string; alsUserInitiated: Boolean; var aHandled: Boolean) — ポップアップ/新しいタブのリクエスト。
- **OnScriptExecuted**(aErrorCode: HRESULT; aResultAsJson: string) — 非同期 JS 結果。
- **OnContentLoading**(alsErrorPage: Boolean) — コンテンツの読み込みが開始されました。
- **OnHistoryChanged** — ナビゲーション履歴が変更されました。
- **OnProcessFailed**(aKind: Integer) — ブラウザプロセスがクラッシュしました。
- **OnPermissionRequested**(aURI: string; aKind: Integer; alsUserInitiated: Boolean; var aState: Integer) — 権限リクエスト。
- **OnWindowCloseRequested** — `window.close()` が呼び出されました。
- **OnDownloadStarting**(aURI, aResultFilePath: string; var aCancel, aHandled: Boolean; var aFilePath: string) — ダウンロードが開始されました。
- **OnDownloadProgress**(aBytesReceived, aTotalBytes: Int64) — ダウンロードの進行状況。
- **OnDownloadCompleted**(aFilePath: string; aState: Integer) — ダウンロード完了。
- **OnContextMenuRequested**(aMenuItems: string; aContextKind: Integer; aLocation: TPoint; var aHandled: Boolean) — 右クリックメニュー。
- **OnFaviconChanged**(aFaviconURI: string) — ページアイコンが変更されました。
- **OnClientCertificateRequested**(aHost: string; aPort: Integer; var aHandled: Boolean) — クライアント証明書が必要です。
- **OnServerCertificateError**(aRequestURI: string; aErrorStatus: Integer; var aAction: Integer) — 証明書エラー。
- **OnStatusBarTextChanged**(aText: string) — ステータスバーのテキストが変更されました。
- **OnDOMContentLoaded** — DOM 準備完了。

- **OnBasicAuthRequested**(aURL: string; var aUserName, aPassword: string; var aHandled: Boolean) — HTTP ベーシック認証。
- **OnCapturePreviewCompleted**(aErrorCode: HRESULT) — スクリーンショット完了。

TsgcWebView2 | ナビゲーション

TsgcWebView2 は URL へのナビゲート、HTML コンテンツの読み込み、ブラウザー履歴の制御のための複数のメソッドを提供します。

基本ナビゲーション

URL を読み込むには **Navigate** メソッドを使用します。コンポーネントはリクエスト前に **OnNavigationStarting** を、完了時に **OnNavigationCompleted** を発生させます。

```
// Navigate to a URL
sgcWebView21.Navigate('https://www.example.com');
```

DefaultURL プロパティを、デザイン時または初期化前に設定することもできます。WebView2の準備ができた後、コンポーネントは自動的にこのURLにナビゲートします。

```
// Set the default URL before initialization
sgcWebView21.DefaultURL := 'https://www.example.com';
```

HTML コンテンツ

NavigateToString を使用して、サーバーやファイルなしに HTML コンテンツを直接ロードします。

```
// Load HTML content directly
sgcWebView21.NavigateToString(
  '<html><body><h1>Hello from Delphi</h1>' +
  '<p>This content was loaded with NavigateToString.</p>' +
  '</body></html>');
```

履歴ナビゲーション

GoBack と **GoForward** を使用してブラウザの履歴をナビゲートします。これらのメソッドを呼び出す前に **CanGoBack** と **CanGoForward** を確認してください。

```
// Navigate back
if sgcWebView21.CanGoBack then
  sgcWebView21.GoBack;

// Navigate forward
if sgcWebView21.CanGoForward then
  sgcWebView21.GoForward;
```

現在のページを更新するには **Reload** を使用し、保留中のナビゲーションをキャンセルするには **Stop** を使用します。

```
// Reload the current page
sgcWebView21.Reload;

// Stop loading
sgcWebView21.Stop;
```

POST ナビゲーション

NavigateWithPostData を使用して、カスタムメソッド、ボディ、ヘッダーで HTTP リクエストを送信します。フォームデータの送信やブラウザーから直接 REST API を呼び出すのに役立ちます。

```
// 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');
```

ナビゲーションイベント

OnNavigationStarting を使用して、ナビゲーションが開始される前に検査またはキャンセルします。リクエストをブロックするには **aCancel** を **True** に設定します。

```
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;
```

ナビゲーションが成功したか失敗したかを確認するには **OnNavigationCompleted** を使用します。

```
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;
```

新しいウィンドウの処理

ページがポップアップやターゲット "_blank" のリンクを開いたとき、**OnNewWindowRequested** イベントが発火します。デフォルトのポップアップを防ぎ、同じウィンドウでナビゲートするには **aHandled** を **True** に設定します。

```
procedure TFormMain.sgcWebView21NewWindowRequested(Sender: TObject;
  const aURI: string; aIsUserInitiated: Boolean;
  var aHandled: Boolean);
begin
  // Navigate in the same window instead of opening a popup
  aHandled := True;
  sgcWebView21.Navigate(aURI);
end;
```


TsgcWebView2 | JavaScript

TsgcWebView2 は、ブラウザコンテキストで JavaScript を実行し、結果を取得し、Delphi アプリケーションと Web コンテンツ間の双方向通信を確立するメソッドを提供します。

非同期実行

ExecuteScript を使用して JavaScript を非同期に実行します。結果は JSON 文字列として **OnScriptExecuted** イベントで返されます。

```
// 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;
```

任意の有効な JavaScript 式を実行できます。戻り値は常に JSON としてシリアル化されます。

```
// Get the number of links on the page
sgcWebView21.ExecuteScript('document.querySelectorAll("a").length');

// Modify page content
sgcWebView21.ExecuteScript(
  'document.body.style.backgroundColor = "lightyellow"');
```

同期実行

結果を直接返すブロッキング実行には**ExecuteScriptSync**を使用してください。値をすぐに必要とする場合はこちらの方がシンプルですが、スクリプトが完了するまで呼び出し元スレッドをブロックします。

```
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

AddInitScript を使用して、ページ上の他のスクリプトより前に、すべてのページロードで自動的に実行される JavaScript を登録します。ポリフィルの注入、ブラウザAPIのオーバーライド、またはメッセージハンドラーの設定に便利です。

```
// 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);' +
  '};');
```

RemoveInitScript を使用して、ID で以前追加された初期化スクリプトの登録を解除します。

Web メッセージング

ウェブメッセージングは Delphi アプリケーションとブラウザで動作する JavaScript の間で双方向通信を提供します。Delphi から JavaScript にデータを送信するには **PostWebMessageAsString** または **PostWebMessageAsJson** を使用し、JavaScript からデータを受信するには **OnWebMessageReceived** を処理してください。

Delphi から JavaScript へのメッセージ送信:

```
// Send a plain string message
sgcWebView21.PostWebMessageAsString('Hello from Delphi!');

// Send a JSON message
sgcWebView21.PostWebMessageAsJson('{"action":"refresh","id":42}');
```

JavaScriptでのメッセージ受信:

```
// 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);' +
  '});');
```

JavaScript から Delphi へのメッセージの受信:

```
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;
```

JavaScript から Delphi にメッセージを送信するには次のようにします:

```
// 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 管理

TsgcWebView2 は、ブラウザのコンテキストで Cookie の読み取り、作成、更新、削除を行うための **CookieManager** プロパティを公開しています。

クッキーの取得

指定した URI のクッキーを取得するには **CookieManager.GetCookies** を使用します。このメソッドは反復できるクッキーオブジェクトのリストを返します。

```
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;
```

クッキーの追加

CookieManager.AddOrUpdateCookie を使用して、新しい Cookie を作成するか既存の Cookie を更新します。最低限、名前、値、ドメイン、パスを指定してください。

```
// 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
```

Cookieの削除

特定の Cookie を名前と URI で削除するには **DeleteCookie** を使用してください。ブラウザからすべての Cookie をクリアするには **DeleteAllCookies** を使用してください。

```
// Delete a specific cookie
sgcWebView21.CookieManager.DeleteCookie(
  'session_id',
  'https://www.example.com');
```

```
// Delete cookies matching a domain and path
sgcWebView21.CookieManager.DeleteCookiesWithDomainAndPath(
  'session_id',
  '.example.com',
  '/');
```

```
// Delete all cookies  
sgcWebView21.CookieManager.DeleteAllCookies;
```

TsgcWebView2 | Download Control

TsgcWebView2 は、ブラウザーが開始したファイルダウンロードをインターセプト、監視、制御するためのイベントを提供します。

ダウンロードイベント

OnDownloadStarting イベントはダウンロードが開始されたときに発火します。ダウンロードをキャンセルしたり、宛先ファイルパスを変更したり、デフォルトの動作でそのまま続行したりできます。

```
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.');  end;  
end;
```

ダウンロードの進行状況

OnDownloadProgress イベントはダウンロード中に定期的に発生し、受信バイト数と予想される合計バイト数を報告します。

```
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;
```

ダウンロード完了

OnDownloadCompleted イベントはダウンロードが完了したときに発火します。ダウンロードが成功したか、キャンセルされたか、失敗したかを判断するには **aState** を確認します。

```
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.');    2: // Failed  
      ShowMessage('Download failed.');  end;  
end;
```

カスタムダウンロードパス

OnDownloadStarting の **aFilePath** パラメータを設定して、ダウンロードをカスタムの場所にリダイレクトします。デフォルトの保存ダイアログを抑制するには **aHandled** を True に設定してください。

```
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 | 設定

TsgcWebView2 はブラウザの動作を制御する TsgcWebView2Settings 型の **Settings** プロパティを公開しています。これらの設定はデザイン時に設定したり、初期化後にランタイムで変更したりできます。

TsgcWebView2Settings プロパティ

- **ScriptEnabled**: Boolean — JavaScript の実行を有効または無効にします。デフォルト: True。
- **WebMessageEnabled**: Boolean — Web メッセージング (postMessage) を有効または無効にします。デフォルト: True。
- **DefaultScriptDialogsEnabled**: Boolean — デフォルトの JavaScript ダイアログ (alert、confirm、prompt) を有効または無効にします。デフォルト値: True。
- **StatusBarEnabled**: Boolean — ブラウザのステータスバーを表示または非表示にします。デフォルト: True。
- **DevToolsEnabled**: Boolean — Edge DevTools へのアクセスを許可または拒否します。デフォルト値: True。
- **ContextMenuEnabled**: ブール値。ブラウザの右クリック コンテキスト メニューを有効または無効にします。デフォルト: True。
- **ZoomControlEnabled**: Boolean — ユーザーが開始するズーム (Ctrl+スクロール、Ctrl+プラス/マイナス) を許可またはブロックします。デフォルト: True。
- **BuiltInErrorPageEnabled**: Boolean — ナビゲーション失敗の組み込みエラーページを表示または非表示にします。デフォルト: True。

デザイン時設定

オブジェクトインスペクターでSettingsプロパティを設定します。これらは、WebView2環境が初期化された後に自動的に適用されます。

```
// 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;
```

ランタイム設定

コンポーネントが初期化された後に実行時に設定を変更できます。変更はすぐに有効になります。

```
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 プロパティは、ブラウザが Cookie、キャッシュ、パーミッション、その他のプロファイルデータを保存するディレクトリを指定します。一意のフォルダーごとに独立したブラウザプロファイルが作成されます。

```
// Use a custom data folder for isolated profiles
sgcWebView21.UserDataFolder := 'C:\AppData\MyApp\Profile1';
```

初期化前（コンポーネントがWebView2環境を作成する前）にこのプロパティを設定します。設定しない場合、アプリケーションの一時ディレクトリにデフォルトのフォルダーが使用されます。

BrowserExecutableFolder

BrowserExecutableFolder プロパティは固定バージョンの WebView2 ランタイムディストリビューションを指します。システムにインストールされたランタイムに依存する代わりに、特定のブラウザーバージョンをアプリケーションに同梱するために使用します。

```
// Use a fixed-version runtime bundled with the application
sgcWebView21.BrowserExecutableFolder :=
  ExtractFilePath(ParamStr(0)) + 'WebView2Runtime';
```

このプロパティは初期化前に設定してください。空のままにすると、コンポーネントはシステムにインストールされた Evergreen WebView2 Runtime を使用します。

TsgcWebView2 | 高度な機能

TsgcWebView2 は、印刷、スクリーンショット、オーディオ制御、証明書処理などの高度なブラウザ機能へのアクセスを提供します。

印刷サポート

PrintToPdf を使用して現在のページを PDF ファイルとして保存するか、**ShowPrintUI** を使用してネイティブの印刷ダイアログを表示します。

```
// Save the current page as PDF
sgcWebView21.PrintToPdf('C:\Output\page.pdf');
```

```
// Show the native print dialog
sgcWebView21.ShowPrintUI;
```

スクリーンショットのキャプチャ

現在のページのスクリーンショットを撮るには **CapturePreviewToFile** を使用してください。format パラメーターで画像タイプを指定します (0 は PNG、1 は JPEG)。キャプチャが完了すると **OnCapturePreviewCompleted** イベントが発生します。

```
// 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;
```

音声 / ミュート制御

音声再生をミュートまたはミュート解除するには、**IsMuted** プロパティを使用します。ページが現在音声を再生しているかどうかを検出するには、**IsDocumentPlayingAudio** を確認します。

```
// Toggle mute
sgcWebView21.IsMuted := not sgcWebView21.IsMuted;

// Check if audio is playing
if sgcWebView21.IsDocumentPlayingAudio then
  LabelStatus.Caption := 'Audio is playing!';
```

証明書の取り扱い

サーバーがクライアント証明書を要求したときに応答するには **OnClientCertificateRequested** を使用します。TLS 証明書エラー (例: 開発中の自己署名証明書) を処理するには **OnServerCertificateError** を使用します。

```
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;
```

バーチャルホストマッピング

SetVirtualHostNameToFolderMappingを使用して、ホスト名をローカルフォルダーにマッピングします。これにより、Webコンテンツはfile://パスの代わりに仮想URLを使用してローカルファイルを参照できます。

```
// 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');
```

プロファイル管理

GetProfileName を使用して現在のブラウザプロファイル名を取得します。**ClearBrowsingData** または **ClearAllBrowsingData** を使用して、キャッシュデータ、クッキー、その他の閲覧アーティファクトを削除します。

```
// Get the profile name
ShowMessage('Profile: ' + sgcWebView21.GetProfileName);

// Clear all browsing data
sgcWebView21.ClearAllBrowsingData;

// Clear specific browsing data types
sgcWebView21.ClearBrowsingData($0001); // cache only
```

基本認証

OnBasicAuthRequested イベントは、サーバーが HTTP Basic 認証を要求したときに発火します。認証情報を提供し、**aHandled** を True に設定します。

```
procedure TFormMain.sgcWebView21BasicAuthRequested(Sender: TObject;
  const aURI: string; var aUserName, aPassword: string;
  var aHandled: Boolean);
begin
  aUserName := 'admin';
  aPassword := 'secret';
  aHandled := True;
end;
```

コンテキストメニュー

OnContextMenuRequested イベントは、ユーザーがブラウザー内で右クリックしたときに発生します。**aHandled** を True に設定すると、デフォルトメニューを抑制して独自のメニューを表示できます。

```
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 プロパティは現在のページのファビコンの URI を返します。**OnFaviconChanged** イベントはファビコンが変更されたときに発生します。

```
procedure TFormMain.sgcWebView21FaviconChanged(Sender: TObject;
  const aFaviconURI: string);
begin
  LabelFavicon.Caption := 'Favicon: ' + aFaviconURI;
end;
```

Status Bar

StatusBarText プロパティは現在のステータスバーテキスト（通常はホバーされたリンクの URL）を返します。**OnStatusBarTextChanged** イベントはテキストが変更されると発生します。

```
procedure TFormMain.sgcWebView21StatusBarTextChanged(Sender: TObject;
  const aText: string);
begin
  StatusBar1.SimpleText := aText;
end;
```

タスクマネージャー

OpenTaskManagerWindow を使用して Edge ブラウザのタスクマネージャーを開きます。これにより各ブラウザプロセスのメモリと CPU 使用量が表示されます。

```
sgcWebView21.OpenTaskManagerWindow;
```

Shared Buffer

高性能なデータ転送のために Delphi アプリケーションと Web コンテンツ間でメモリバッファを共有するには **PostSharedBufferToScript** を使用します。

```
// 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}');
```

ダイレクトCOMアクセス

コンポーネント API でカバーされていない高度なシナリオには、**WebView**、**Controller**、**Environment** プロパティを使用して基盤となる WebView2 COM インターフェースに直接アクセスします。

```
var
  vWebView: ICoreWebView2;
begin
  vWebView := sgcWebView21.WebView;
  if Assigned(vWebView) then
  begin
    // Call any ICoreWebView2 method directly
```

```
end;  
end;
```

TsgcUDPCClient

TsgcUDPCClient は *Indy* ライブラリをベースにした UDP クライアントを実装します。

はじめに

TsgcUDPCClient は Indy ライブラリをベースにした UDP クライアントを実装します。

UDP はコネクションレス型プロトコルで、送信されたメッセージが宛先に到達することの保証はありませんが、TCP プロトコルとは対照的にはるかに高速です。

1. フォームに **TsgcUDPCClient** コンポーネントをドロップします
2. **Host** と **Port**（デフォルトは 80）を設定して、利用可能な UDP サーバーに接続します。

```
oClient = new TsgcUDPCClient();  
oClient->Host = "127.0.0.1";  
oClient->Port = 80;
```

3. HTTPプロキシサーバーを通じて接続することができます。プロキシのプロパティを定義する必要があります:

Host: プロキシサーバーのホスト名。

Port: プロキシサーバーのポート番号。

Username: 認証用ユーザー名。匿名の場合は空白のままにします。

パスワード: 認証用のパスワード。匿名の場合は空白のままにしてください。

4. 必要に応じてイベントを処理できます
5. **WriteData**メソッドを呼び出して、任意のメッセージをUDPサーバーに送信します。

リファレンス

- プロパティ — コンポーネントで使用可能な 9 個のプロパティ。
- メソッド — 2 つのパブリックメソッド。
- イベント — 処理できる3個のイベント。

TsgcUDPServer

TsgcUDPServer は *Indy* ライブラリに基づいたUDPサーバーを実装します。

はじめに

TsgcUDPServer は Indy ライブラリに基づいたUDPサーバーを実装します。

UDP は接続レスプロトコルで、送信されたメッセージが宛先に届くという保証はありませんが、TCP プロトコルとは逆に、はるかに高速です。

1. フォームに **TsgcUDPServer** コンポーネントをドロップします。
2. リスニング **Port** を設定します。

```
oClient = new TsgcUDPServer();  
oClient->Port = 80;
```

3. サーバーを**起動**するには、プロパティ **Active = true** を設定します。

リファレンス

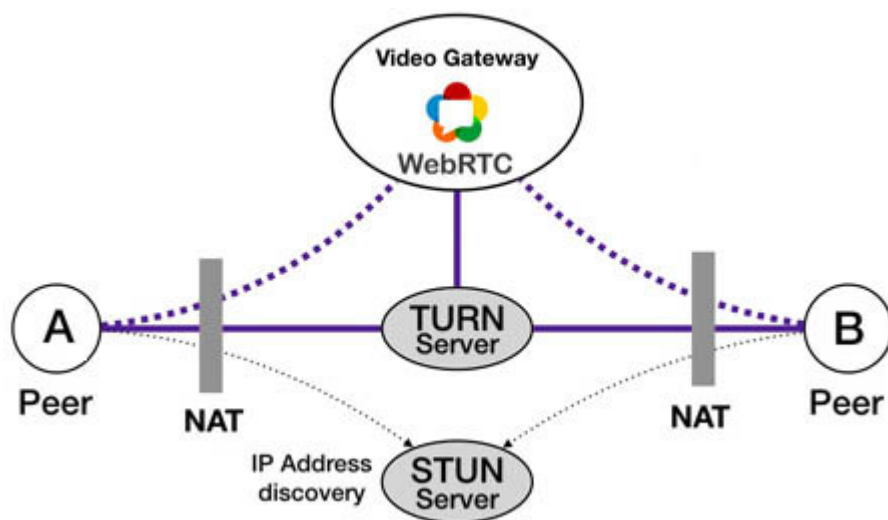
- プロパティ — コンポーネントで利用可能なプロパティ 11 件。
- メソッド — 公開メソッド 7 件。
- イベント — 処理できるイベント6件。

STUN

STUN (Session Traversal Utilities for NAT) はIPネットワークにおけるリアルタイム音声およびビデオのために使用されるIETFプロトコルです。STUNはサーバー・クライアントプロトコルで、STUNサーバーは通常UDPとTCPの両方で動作し、ポート3478でリスンします。

STUN プロトコルの主な目的は、NAT の背後にあるデバイスが自身のパブリック IP と NAT の種類を検出できるようにすることです。

STUN は NAT の背後にいるピア間で通信するためのメカニズムを提供します。ピアは STUN サーバーにリクエストを送信して、公開 IP アドレスとポートを確認します。クライアントからサーバーに送信されるバインディングリクエストは、NAT が割り当てた IP とポートのバインディングを確認するために使用されます。STUN クライアントは STUN サーバーにバインディングリクエストを送信し、サーバーはクライアントが使用するソース IP とポートを調べて、この情報をクライアントに返します。



STUNサーバーは基本的に、成功またはエラーの2種類の応答を送信します。すべての応答には、バインディングのIPアドレス、エラーコード、エラーの理由などの情報を含む属性のリストがあります。

コンポーネント

- **TsgcSTUNClient:** STUN プロトコルを実装し、STUN サーバーにバインディングリクエストを送信できるクライアントコンポーネント。
- **TsgcSTUNServer:** STUNプロトコルを実装する サーバーコンポーネント。

TsgcSTUNClient

TsgcSTUNClient コンポーネント。

はじめに

TsgcSTUNClientは、STUNプロトコルを実装するクライアントであり、STUNサーバーへbinding要求を送信できます。

このコンポーネントでは **UDP** と **TCP** をトランスポートとして使用できます。UDP をトランスポートとして使用する場合、短時間後にレスポンスが届かないときにリクエストを再送する**再送メカニズム**を実装しています。

基本的な使い方

通常、STUNサーバーはUDPポート3478で実行され、認証を必要としません。そのため、STUN request bindingを送信するには、クライアントが接続先を知ることができるようサーバープロパティを設定し、コンポーネントがサーバーから応答を受信するイベントを処理してください。

サーバーの設定

- Host: サーバーの IP または DNS 名 (例: stun.sgcwebsockets.com)。
- Port: リスニングサーバーポート。例: 3478

SendRequest メソッドを呼び出して、STUN サーバーへのリクエストバインディングを送信します。

イベントの処理

- サーバーが成功応答を返した場合、イベント **OnSTUNResponseSuccess** が呼び出され、**aBinding** オブジェクトを読み取ることでバインディング情報にアクセスできます。
- サーバーがエラーを返した場合、**OnSTUNResponseError** イベントが呼び出され、**aError** オブジェクトを読み取ることでエラーコードと理由にアクセスできます。

```
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);
}
```

リファレンス

- プロパティ — コンポーネントで利用可能なプロパティ 9 件。
- メソッド — 3 つのパブリックメソッド。
- イベント — 処理可能なイベントが 4 つあります。

ガイド

- バインディング
 - [UDP 再送信](#)
 - [長期的な資格情報](#)
- [属性](#)

STUN クライアント | UDP 再送信

UDP 上で STUN を実行している場合、STUNメッセージがネットワークによってドロップされる可能性があります。STUNリクエスト/レスポンスランザクションの信頼性は、クライアントアプリケーション自身によるリクエストメッセージの再送信によって実現されます。

クライアントは STUN リクエストメッセージを RTO（「再送信タイムアウト」）の間隔で再送信を開始し、再送信ごとに倍増させます。RTO はラウンドトリップ時間の推定値です。

デフォルトでは、sgcWebSockets の STUN クライアントは RTO として 500 ms、最大再試行回数として 7 が設定されています。

たとえば RTO が 500 ms の場合、リクエストは 0 ms、500 ms、1500 ms、3500 ms、7500 ms、15500 ms、31500 ms の時刻に送信されます。クライアントが 39500 ms 後もレスポンスを受信していない場合、ランザクションはタイムアウトしたと見なされます。

```
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クライアント | 長期資格情報

長期資格情報メカニズムは、クライアントとサーバー間で共有されるユーザー名とパスワードの形式の長期資格情報に依存しています。この資格情報は、ユーザーにプロビジョニングされ、ユーザーがシステムのサブスクライバーでなくなるかまたは変更されるまで有効であると想定されているため、長期とみなされます。

以下のコードを使用して、sgcWebSockets STUN クライアントで長期資格情報を設定できます。

```
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();
```

サーバーが長期資格情報を要求し、クライアントが送信した資格情報が誤っている場合、クライアントは **OnSTUNResponseError** イベントで 401 Unauthorized エラーレスポンスを受け取ります。

STUN クライアント | 属性

サーバーがリクエストバインディングへのレスポンスメッセージとしてクライアントにメッセージを送信するたびに、STUN メッセージにはレスポンスに関する情報を含む属性のリストが含まれます。

TsgcSTUN_Message クラスとその Attributes プロパティを使用してこれらの属性にアクセスできます。Attributes プロパティには、有用な情報を持つ TsgcSTUN_Attribute オブジェクトのリストが含まれています。

```
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 サーバーは認証の有無にかかわらず設定でき、*Fingerprint* 属性の検証、代替サーバーの送信など多くの機能があります。

はじめに

TsgcSTUNServerは、STUNプロトコルを実装し、STUNクライアントからのバインディングリクエストを処理できるサーバーです。

STUN サーバーは認証の有無にかかわらず設定でき、*Fingerprint* 属性の検証、代替サーバーの送信など多くの機能があります。

基本的な使い方

通常 STUN サーバーは UDP ポート 3478 で動作し認証は不要です。STUN サーバーを設定するには、リスニングポート (デフォルト 3478) を設定してサーバーを起動します。

サーバーの設定

- Port: リスニングサーバーポート。例: 3478

STUN サーバーを起動するには、プロパティ **Active = True** を設定します。

```
TsgcSTUNServer oSTUN = new TsgcSTUNServer();
oSTUN->Port = 3478;
oSTUN->Active = true;
```

リファレンス

- プロパティ — コンポーネントで利用可能な 8 つのプロパティ。
- メソッド — 2 つのパブリックメソッド。
- イベント — 処理できる 4 つのイベント。

ガイド

- 設定
 - [長期資格情報](#)
 - [Alternate Server](#)

STUN Server | Long-Term Credentials

通常、STUN サーバーは認証なしで設定されているため、任意の STUN クライアントがバインディングリクエストを送信し、認証なしでサーバーからのレスポンスを期待できます。

sgcWebSockets STUN サーバーは長期認証情報をサポートしているため、長期認証情報を含むバインディングリクエストのみを許可するように TsgcSTUNServer を設定できます。

設定するには、STUNOptions.Authorization プロパティにアクセスして有効にしてください。次に LongTermCredentials プロパティにアクセスして有効にします。デフォルトでは、この種類の認証は Realm 文字列がすでに設定されており、デフォルトの StaleNonce 値は10分（600秒）です。

```
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 サーバー | 代替サーバー

代替サーバーは、STUN クライアントが試みるべき別の STUN サーバーを識別する代替トランスポート アドレスを表します。

STUNサーバーは、バインディングリクエストへのレスポンスとして代替サーバーを送信するように設定できます。この動作を設定するには、STUNOptions.BindingAttributes.AlternateServer プロパティにアクセスし、必要な値をここで設定するだけです。

```
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;
```

クライアントが Alternate Server レスポンス属性を受け取ると、新しいサーバーへのバインディングリクエストを送信しようとします。

TURN

Traversal Using Relays around NAT (TURN) プロトコルを使用すると、サーバーがデバイス間でデータパケットを中継できます。

発信者と着信者の両方のパブリック IP アドレスが検出できない場合、TURN はエンドポイント間の通話の中継するフォールバック技術を提供します。

WebRTC セッションの接続は、複数の WebRTC サーバーの支援を受けて調整された取り組みで行われます。WebRTC における NAT トラバーサルサーバーはメディアが正しく接続されることを確実にする役割を担います。これらのサーバーは STUN と TURN です。

WebRTC セッションの接続方法

直接

両方のデバイスがローカルネットワーク上にある場合、相互に接続するための特別な作業は不要です。一方のデバイスが他方のデバイスのローカルIPアドレスを持っていれば、直接通信できます。

パブリック IP アドレスで直接接続

STUN プロトコルで取得した公開 IP アドレスを使用して WebRTC を直接接続します。

TURN サーバーを経由したルーティング

ピアが NAT の背後にあり、ファイアウォールが存在する場合、直接接続はできないため、ピア間のデータをルーティングするために TURN サーバーが必要です。

コンポーネント

- **TsgcTURNClient:** TURN プロトコルを実装し、TURN サーバーへのアロケーション、パーミッションの作成、インディケーションの送信などを行うクライアントコンポーネントです。
- **TsgcTURNServer:** TURN プロトコルを実装するサーバーコンポーネント。

TsgcTURNClient

TsgcTURNClient コンポーネント。

はじめに

TsgcTURNClient は TURN プロトコルを実装し、TURN サーバーへのアロケーションリクエストを送信できるクライアントです。クライアントは STUN クライアントから継承しているため、[STUN クライアント](#) がサポートするすべてのメソッドは TURN クライアントでも既にサポートされています。

基本的な使い方

通常、TURN サーバーは UDP ポート 3478 で動作し、認証は不要です。TURN リクエストを送信するには、クライアントが接続先を知り、サーバーからのレスポンスをコンポーネントが受信するイベントを処理できるよう、サーバープロパティを設定してください。

サーバーの設定

- Host: サーバーの IP または DNS 名。例: turn.sgcwebsockets.com
- Port: リスニングサーバーポート。例: 3478

Allocate メソッドを呼び出して、TURNサーバーにIPアドレスとポートを割り当てるリクエストを送信します。

イベントの処理

- サーバーが成功した応答を返すと、**OnTURNAllocateSuccess** イベントが呼び出され、**aAllocation** オブジェクトを読み取ってアロケーション情報にアクセスできます。
- サーバーがエラーを返した場合、**OnSTUNResponseError** イベントが呼び出され、**aError** オブジェクトを読み取ることでエラーコードと理由にアクセスできます。

```
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);
}
```

リファレンス

- プロパティ — コンポーネントで使用可能な 10 個のプロパティ。
- メソッド — 9個のpublicメソッド。
- イベント — 処理できる 11 のイベント。

ガイド

- アロケーション

- [IPアドレスの割り当て](#)
- [パーミッションの作成](#)
- [インジケーション](#)
 - [インジケーションの送信](#)
- [チャンネル](#)
 - [TURNクライアントチャンネル](#)

TURN クライアント | IP アドレスのアロケーション

TURN プロトコルを使用すると、NAT 内にいるピア間でデータを交換するためにリレー IP アドレスを使用できます。

TURNサーバーで新しいRelayed IP Addressを作成するには、クライアントはまず**Allocate**メソッドを呼び出す必要があります。このメソッドは、新しいRelayed IP Addressを作成するためのRequestをTURNサーバーに送信します。TURNサーバーが新しいRelayed IP Addressを作成できる場合、クライアントは成功応答を受信します。クライアントは、Allocationのlifetimeで定義された時間中、他のピアと通信できます。

```
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);
}
```

有効期限切れを防ぐために、**Refresh** メソッドを使用してライフタイムを更新できます。ライフタイムは有効期限までの秒数です。値がゼロの場合、アロケーションは削除されます。

```
oTURN->Refresh(600);
```

TURNクライアント | パーミッションの作成

TURN サーバーで新しいアロケーションが作成された場合、そのアロケーションはパーミッションが設定されていないと他のピアからの受信パケットを処理できません。したがって、他のピアがリレー IP アドレスを使用して通信できるようにするには、まず TURN クライアントがデータの交換を許可する IP アドレスのパーミッションを作成する必要があります。

新しいパーミッションを作成するには、**CreatePermission** メソッドを呼び出し、ピアの IP アドレスをパラメーターとして渡してください。ピア IP が TURN サーバーにすでに存在する場合は更新されます。存在しない場合は作成されます。パーミッションは更新されない限り 5 分後に期限切れになります。

TURN クライアントは、アクティブなアロケーションが存在する場合にのみ **CreatePermission** メソッドを呼び出すことができます。

パーミッションの作成に成功すると、イベント **OnTURNCreatePermission** が呼び出されます。

```
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 クライアント | インジケーションの送信

TURN プロトコルは、ピアとのデータ送受信に 2 つのメカニズムをサポートしています。そのうちの 1 つは Send および Data メカニズムです。

TURN クライアントは **SendIndication** メソッドを使用して、ピアへの中継のためにサーバーにデータを送信できます。TURN クライアントは、Send Indication が送信されるピアの IP アドレスに対する許可が設定されていることを確認する必要があります。

SendIndication メソッドへのレスポンスは **OnTURNDDataIndication** イベントで受信されます。

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

void OnTURNDDataIndication(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 クライアント | チャンネル

チャンネルは TURN クライアントとサーバーが `ChannelData` メッセージを使用してアプリケーションデータを送受信する方法を提供します。これは [Send](#) および [Data](#) インジケーションよりもオーバーヘッドが少ないです。

`ChannelData` メッセージを使用してピア間でデータを交換する前に、TURN クライアントは新しいチャンネルを作成する必要があります。これを行うには、ピアの IP アドレスとポートをパラメーターとして渡して **`ChannelBind`** メソッドを呼び出してください。

TURNサーバーが新しいチャンネルをバインドできる場合、TURNクライアントは成功応答**`OnTURNChannelBind`** イベントを受信します。

```
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));
}
```

チャンネルバインディングは、リフレッシュされない限り10分間持続します。チャンネルをリフレッシュするには、**`ChannelBind`**メソッドを再度呼び出すだけです。

TURN クライアントが新しいチャンネル メッセージを受信すると、**`OnTURNChannelData`** イベントが呼び出されます。

```
void OnTURNChannelData(TObject *Sender, const TsgcSocketConnection *aSocket,
    const TsgcTURNChannelData *aChannelData)
{
    DoLog("#Channel Data: [" + IntToStr(aChannelData->ChannelID) + "] " +
        sgcGetStringFromBytes(aChannelData->Data));
}
```

TsgcTURNServer

TURN サーバーはロングターム認証、Allocation、パーミッション、チャンネルデータなどをサポートします。

はじめに

TsgcTURNServerはTURNプロトコルを実装し、TURNクライアントからのリクエストを処理できるサーバーです。このコンポーネントは **TsgcSTUNServer** から継承しているため、すべてのメソッドとプロパティが **TsgcTURNServer** で利用できます。

TURN サーバーはロングターム認証、Allocation、パーミッション、チャンネルデータなどをサポートします。

基本的な使い方

通常、TURN サーバーは UDP ポート 3478 で動作し、長期資格情報を必要とします。TURN サーバーを設定するには、リスンポート（デフォルト 3478）を設定してサーバーを起動します。

サーバーの設定

- Port: リスニングサーバーポート。例: 3478
- `TURNOptions.Authentication.LongTermCredentials` に長期認証情報プロパティを定義します。
- TURN クライアントが TURN サーバーにリクエストを送信するときにパスワードを設定するために `OnSTUNRequestAuthorization` を処理してください。

TURN サーバーを起動するには、プロパティ **Active = True** を設定します。

```
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";
    }
}
```

リファレンス

- プロパティ — コンポーネントで使用可能な 9 つのプロパティ。
- メソッド — 2 つのパブリックメソッド。
- イベント — 処理できるイベント 11 件。

ガイド

- 設定
 - [長期資格情報](#)
- アロケーション
 - [Allocation](#)

TURN Server | Long Term Credentials

通常、TURN サーバーは TURN リクエストには認証あり、STUN リクエストには認証なしで設定されます。

sgcWebSockets TURN サーバーは長期資格情報をサポートしているため、TsgcTURNServer を構成して長期資格情報情報を持つリクエストのみを許可することができます。

設定するには、TURNOptions.Authorization プロパティにアクセスして有効にします。次に LongTermCredentials プロパティにアクセスして有効にします。デフォルトでは、この種類の認証は Realm 文字列がすでに設定されており、デフォルトの StaleNonce 値は10分（600秒）です。

```
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 サーバー | アロケーション

すべてのTURN操作はアロケーションを中心に展開され、すべてのTURNメッセージはアロケーションに関連付けられています。アロケーションは以下で構成されます。

- 中継されたトランスポートアドレス
- 5タプル：クライアントの IP アドレス、クライアントの IP ポート、サーバーの IP アドレス、サーバーポート、トランスポートプロトコル。
- 認証情報。
- 各リレートランスポートアドレスの有効期限。
- 各リレーされたトランスポートアドレスの権限リスト。
- 各中継されたトランスポートアドレスのチャンネルバインディングリスト。

TURN クライアントが Allocate リクエストを送信すると、この TURN メッセージはサーバーで処理され、新しいリレードトランスポートアドレスの作成が試みられます。デフォルトでは、利用可能な UDP ポートがあれば新しいリレードアドレスが作成されますが、**OnTURNBeforeAllocate** イベントを使用して新しい Allocation リクエストを拒否することができます。

```
void OnTURNBeforeAllocate(TObject *Sender, const TsgcSocketConnection *aSocket,
    const string aIP, Word aPort, ref bool Reject)
{
    if (your own rules) == false
    {
        Reject = false;
    }
}
```

プロセスが実行されると、サーバーは新しい Allocation を作成し、**OnTURNCreateAllocation** イベントが呼び出されます。このイベントは TsgcTURNAllocationItem クラスを通じて Allocation に関する情報を提供します。

```
void OnTURNCreateAllocation(TObject *Sender, const TsgcSocketConnection *aSocket,
    const TsgcTURNAllocationItem *Allocation)
{
    DoLog("New Allocation: " + Allocation->RelayIP + ":" + IntToStr(Allocation->RelayPort));
}
```

アロケーションの有効期限が切れるか、またはクライアントから lifetime に 0 を指定したリフレッシュリクエストを受信して削除された場合、**OnTURNDeleteAllocation** イベントが発火します。

```
void OnTURNDeleteAllocation(TObject *Sender, const TsgcSocketConnection *aSocket,
    const TsgcTURNAllocationItem *Allocation)
{
    DoLog("Allocation Deleted: " + Allocation->RelayIP + ":" + IntToStr(Allocation->RelayPort));
}
```

ICE

Interactive Connectivity Establishment (ICE) プロトコルは NAT トラバーサルに使用されます。ICE は Session Traversal Utility for NAT (STUN) および Traversal Using Relay NAT (TURN) などの方法を組み合わせて使用します。Network Address Translator (NAT) の存在は Voice over IP (VoIP) および WebRTC の実装に問題をもたらします。

コンポーネント

- **TsgcICEClient:** ICEプロトコルを実装し、候補の取得、交換、検証を可能にするクライアントコンポーネント。

TsgcICEClient

ICEOptions。

はじめに

TsgcICEClient は ICE プロトコルを実装し、TURN サーバーへのアロケーションリクエストを送信できるクライアントです。このクライアントには [TsgcTURNClient](#) と [TsgcWebSocketClient](#) が必要です。

設定

ICE クライアントには次のプロパティがあります。

- **ICEOptions.CheckList.MaxCandidates:** クライアントが処理できる候補の最大数です。デフォルトは 100 です。
- **ICEOptions.GatherCandidates.STUN:** デフォルトは true で、STUN プロトコルを使用して候補（再帰アドレス）を取得します。
- **ICEOptions.GatherCandidates.TURN:** デフォルトは True。TURN プロトコルを使用して候補（リレーアドレス）を取得します。

ICE クライアントは候補を収集するために **TURN クライアント**が必要です。**GatherCandidates** メソッドを呼び出す前に [TsgcTURNClient](#) を [TsgcICEClient](#) の TURN プロパティにリンクしてください。[TURN クライアント](#)の設定に関する詳細情報を取得できます。

```
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();
```

リファレンス

- プロパティ — コンポーネントで利用可能な3個のプロパティ。
- メソッド — 3個のパブリックメソッド。
- イベント — 処理可能な 6 個のイベント。

ガイド

- 候補
 - [候補の収集](#)
 - [候補のペアリング](#)

ICE | 候補の収集

ICE は候補の収集を開始し、通常ローカル IP アドレス、STUN プロトコルを使用したリフレクシブアドレス、TURN プロトコルを使用したリレーアドレスを取得します。

収集を開始するには **GatherCandidates** メソッドを呼び出します。これにより内部タイマーが起動し、まずローカル IP アドレスを取得し、次に STUN サーバーに接続して reflexive IP アドレスを取得し、最後に TURN サーバーに接続して relayed IP アドレスを取得します。

新しい候補が取得されるたびに、**OnICECandidate** イベントが非同期に呼び出されます。候補の収集中にエラーが発生した場合は、**OnICECandidateError** イベントがトリガーされます。

```
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 | 候補ペア

候補（ローカルおよびリモート）が取得され、SDPの説明が設定されたら、ICE 呼び出し側クライアントはすべての候補ペアを処理してデータ交換可能なものを探し始めます。このプロセスを開始するには、メソッド **ProcessCandidates** を呼び出します。

ProcessCandidates メソッドはすべてのペア候補を評価し、STUNバインディングパケットを送信します。STUNバインディングパケットが相手ピアから応答として受信された場合、その2つのピア間で接続が可能であることを意味するため、そのペアが指定されます。

ペアリングが成功した場合、**OnICECandidatePairNominated** イベントが非同期的にトリガーされます。ペアリングにエラーがあるかタイムアウト後に接続できない場合、**OnICECandidatePairFailed** イベントがトリガーされます。

TsgcRTCPeerConnection

TsgcRTCPeerConnection は、UDP を通じて P2P でピアを接続できるクライアントコンポーネントです。

はじめに

[*このコンポーネントは現在開発中です]

TsgcRTCPeerConnection は P2P を介して UDP でピアを接続できるクライアントコンポーネントです。フローは 4 つのステップに分けられます。

- シグナリング
- 接続中
- セキュリティ保護
- 通信

これらのステップを実装するために、クライアントは次のプロトコルを使用します:

- **WebSocket:** このプロトコルはシグナリングに使用され、クライアントはセッション記述プロトコルおよびローカル・パブリック・リレーIPアドレスを交換します。
- **UDP:** これはトランスポートプロトコルです。クライアントは UDP を使用してピア間でメッセージを送受信します。
- **DTLS:** TLS に似た暗号化仕様で、ピア間のメッセージを保護し、第三者がメッセージを読み書きできないようにします。
- **STUN:** パブリックIPアドレスを取得するためのプロトコル。
- **TURN:** NAT の背後にあるピアの IP アドレスを中継するプロトコル。
- **ICE:** ピア間でアクセス可能な IP アドレスとポートを見つけるためのプロトコル。

シグナリング

クライアントが起動した時点では、通信相手や通信内容についての情報は一切ありません。シグナリングは、以下のような詳細情報を含む SDP (Session Description Protocol) を使用します。

- ピアが到達可能な IP とポート。
- 通信を保護するために使用される、Fingerprintの証明書です。
- User and Password.
- ...

シグナリングはデータ交換に WebSocket プロトコルを使用し、サブプロトコルを通じて機能し、サーバー側の TsgcWSPServer_RTCPeerConnection コンポーネントに実装されています。TsgcRTCPeerConnection はすでに TsgcWSPClient_RTCPeerConnection が添付された WebSocket クライアントを内部で作成します。

IPとポートを取得するために、クライアントはSTUN/TURNプロトコルを使用してこの情報を取得します。そのため、STUN/TURNサーバーも必要です。

リンク:

- [RTCPeerConnection WebSocket サーバー](#)
- [RTCPeerConnection WebSocket クライアント](#)
- [RTCPeerConnection STUN TURN](#)
- [RTCPeerConnection シグナリング](#)

接続中

2つのピアが候補とSDPを把握したら、クライアントはICEと呼ばれる別の標準プロトコルを使用します。

ICE (Interactive Connection Establishment) は、2 つのピア間の接続確立を可能にします。ピアは同じネットワーク内にあることも、NAT の背後にあることもあります。ICE は中央サーバーなしに直接接続を確立するためのソリューションです。P2P 接続が不可能な場合、ICE は TURN サーバーを使用してデータをリレーするために TURN を使用します。

ICE が 2 つのピア間で接続できる有効な候補を見つけた後、次のステップは通信を暗号化することです。

リンク:

- [RTCPeerConnection ICE](#)

セキュリティ保護

ピアが接続した後、通信はセキュアでなければなりません。これは UDP 上の通信を保護するために使用される暗号化プロトコルである DTLS を使用して行われます。

DTLS ハンドシェイクが正常に処理されると、別のプロトコルである SRTP (Secure Real-Time Transport Protocol) が使用されますが、現在 SRTP は実装されていません。

リンク:

- [RTCPeerConnection DTLS](#)

通信

2 つのピアがセキュアプロトコルを使用すると、通信は 2 つのプロトコルを使用して行われます:

- **RTP**: Real Time Transport Protocol (リアルタイムトランスポートプロトコル): SRTP で暗号化されたメディアを交換するために使用されます。
- **SCTP**: ストリーム制御伝送プロトコル。dTLSで暗号化されたデータチャンネルメッセージの送受信に使用されます。

現在、これらのプロトコルは実装されていませんが、UDP 経由の DTLS を使用してデータを送受信できます。

リンク:

- [RTCPeerConnection データ](#)

リファレンス

- プロパティ — コンポーネントで使用可能な 2 つのプロパティ。
- メソッド — 3個のパブリックメソッド。
- イベント — 処理できる 14 のイベント。

RTCPeerConnection | WebSocket サーバー

TsgcRTCPeerConnection クライアントにはシグナリング用の WebSocket サーバーが必要です。クライアントは WebSocket プロトコルを使用してピアの SDP と候補（IP とポート）を交換します。これによりピアが通信できるようになります。

WebSocketサーバーを設定するには、sgcWebSocketsライブラリでWebSocketサーバー利用可能のいずれかを使用し、**TsgcWSPServer_RTCPeerConnection** をアタッチできます。これはRTCPeerConnectionで使用されるサブプロトコルです。

```
TsgcWebSocketServer *oServer = new TsgcWebSocketServer();
TsgcWSPServer_RTCPeerConnection *oProtocol = new TsgcWSPServer_RTCPeerConnection();
oProtocol->Server = oServer;
oServer->Port = 8080;
oServer->Active = true;
```

新しい WebSocket クライアントが WebSocket サーバーに接続するたびに、サーバーは同じチャンネルでリスンしている他のピアがあるかどうかを確認し、それに応じてデータを転送します。

RTCPeerConnection | WebSocket クライアント

RTCPeerConnection は内部的にカスタムサブプロトコルを持つ WebSocket クライアントを作成し、WebSocket サーバーと通信します。RTCOptions.WebSocket プロパティに WebSocket 接続を定義する値があります。

- **Host:** サーバーの DNS または IP アドレス。例: 127.0.0.1 または www.esegce.com。
- **Port:** WebSocket サーバーのリッスンポート。
- **TLS:** サーバーがセキュア接続を使用している場合は有効にします。
- **Channel:** ピア間でデータを交換するために使用されるチャンネル名（両方のピアが同じチャンネル名を持つ必要があり、最大ピア数は 2）。

RTCPeerConnection | STUN TURN

TsgcRTCPeerConnection はパブリック IP アドレスとリレーされた IP アドレス（必要な場合）を取得するために STUN/TURN プロトコルを使用します。そのため、この情報を取得するための STUN/TURN サーバーが必要です。STUN/TURN サーバーの詳細は次のリンクを参照してください: [TURN サーバー](#)。

STUN/TURNサーバーが稼働したら、TsgcRTCPeerConnectionのRTCOptions.ICEプロパティでTURN Serverプロパティを構成できます。

- **Host:** TURNサーバーのIPアドレスまたはDNS。例: 127.0.0.1 または www.esegece.com。
- **Port:** 通常はデフォルトポート3478です。
- **Username:** TURN サーバーが長期認証情報を使用している場合のユーザー名 (デフォルト)。
- **Password:** TURNサーバーがLong-term資格情報(デフォルト)を使用している場合のパスワード。

RTCPeerConnection | シグナリング

TsgcRTCPeerConnection が RTCOptions プロパティを設定し、サーバー（WebSocket および STUN/TURN）が実行されると、クライアントは候補の収集プロセスを開始できます。

クライアントは最初に WebSocket サーバーに接続し、接続が成功するとローカル SDP を送信します。次にローカルと公開の IP アドレスを取得しようとします。公開 IP アドレスを取得するために STUN サーバーにバインディングリクエストを送信し、公開 IP アドレスと TURN サーバーのリレー IP アドレスを取得します。新しい候補を取得するたびに、この情報は WebSocket サーバーに渡され、相手ピアに転送されます。

RTCPeerConnection がローカル SDP、リモート SDP、およびキャンディデートを持つと、各キャンディデートペアが接続できるかどうかを確認するプロセスを開始します。キャンディデートペアが正常に接続されると、有効なキャンディデートペアとなり、セキュリティプロセスがフローを継続します。

RTCPeerConnection | ICE

ICE (Interactive Connectivity Establishment) は 2 つのピアを接続するために使用されるプロトコルです。2 つのピア間のすべての可能なルートを決出し、接続されていることを確認します。これらのルートはキャンディデートペアとも呼ばれ、ローカルとリモートのトランスポートアドレスのペアリングです。これらのアドレスはローカル IP アドレス、パブリック IP アドレス、またはリレートランスポートアドレスになります。各ピアは使用したいすべてのアドレスを収集し、それらを交換してから接続を試みます。

アドレスの収集

アドレス収集時に呼び出されるイベントを以下に示します。

OnRTCLocalCandidate

このイベントは新しいローカル候補が見つかったときに呼び出されます。

OnRTCRemoteCandidate

このイベントは、WebSocketサーバーがこのピアへリモート候補を送信したときに呼び出されます。

OnRTCLocalDescription

このイベントは TsgcRTCPeerConnection がローカル SDP を必要とするときに呼び出されます

OnRTCRemoteDescription

WebSocket サーバーがリモート SDP をこのピアに送信したときにイベントが呼び出されます。

接続性テスト

ピアが他のピアに接続できるかをテストするためにバインディングリクエストを送信するとき、次のイベントが呼び出される場合があります。

OnRTCCandidatePairNominated

両方のピアがこの候補ペアを使用して接続できる場合、イベントが呼び出されます。

OnRTCCandidatePairFailed

ピアがこの候補ペアを使用して接続できない場合、このイベントが呼び出されます。

OnRTCConnect

このイベントは有効な候補ペアが存在し、DTLS が有効でない場合に呼び出されます。DTLS が有効な場合、このイベントは DTLS ハンドシェイクが成功した後に呼び出されます。

RTCPeerConnection | DTLS

有効な候補ペア（両ピアが接続してデータを交換できる状態）が見つかったら、接続をセキュアにする段階です。DTLS は暗号化プロトコルで、データを暗号化して交換される内容の傍受や改ざんを防ぎます。

DTLS には **openssl ライブラリ（openssl 1.1 以降）が必要です。**

DTLS の設定は `TsgcRTCPeerConnection` の `RTCOptions.DTLSOptions` プロパティにあります。DTLS を有効にするには `RTCOptions.DTLS` プロパティを `True` に設定します。以下に主なプロパティを示します。

RootCertFile: ルート証明書ファイルへのパス。

CertFile: 証明書ファイルへのパス。

KeyFile: 証明書鍵ファイルへのパス。

VerifyCertificate: 証明書を検証する必要がある場合は、このプロパティを有効にしてください。

VerifyDepth: 検証時に許可されるリンクの最大数を表す整数プロパティです。

OpenSSL_Options: OpenSSL ライブラリの設定。

APIVersion: 使用する OpenSSL API を定義できます。DTLS をサポートするのは OpenSSL API 1.1 以降のみです。

oslAPI_1_1: API 1.1 の OpenSSL を使用します。カスタム Indy ライブラリが必要で、OpenSSL 1.1.1 ライブラリ（TLS 1.3 サポート付き）の使用が可能になります。

oslAPI_3_0: API 3.0 OpenSSL を使用します。当社のカスタム Indy ライブラリが必要であり、OpenSSL 3.0.0 ライブラリを使用を可能にします（TLS 1.3 サポート付き）。

LibPath: openssl ライブラリの場所を設定できます

oslpNone: これはデフォルトです。openssl ライブラリは、バイナリと同じフォルダー、または既知のパスにある必要があります。

oslpDefaultFolder: すべての IDE パーソナリティに対してライブラリが配置されるべき openssl パスを自動的に設定します。

oslpCustomFolder: このオプションが選択されている場合、プロパティ `LibPathCustom` にフルパスを定義します。

LibPathCustom: `LibPath = oslpCustomFolder` の場合、openssl ライブラリが格納されているフォルダーの完全パスをここで定義します。

UnixSymLinks: Unix システムでシンボリックリンクの読み込みを有効または無効にします（デフォルトでは有効です。OSX64 は除きます）。

oslsSymLinksDefault: デフォルトでは有効です。ただし OSX64 では無効です（MacOS Monterey 以降、バージョンなしでライブラリを読み込もうとすると失敗するため）。

oslsSymLinksLoadFirst: `SymLinks` を読み込み、バージョンライブラリを読み込む前に行います。

oslsSymLinksLoad: バージョンライブラリの読み込みを試みた後にシンボリックリンクを読み込みます。

oslsSymLinksDontLoad: `SymLinks` を読み込みません。

RTCPeerConnection | データ

接続が成功したら、**WriteData** メソッドを使用して相手のピアにメッセージを送信できます。接続は DTLS over UDP を使用し、可能であれば P2P 接続になります（P2P が不可能な場合はリレーアドレスを使用）。

データの送信

WriteData メソッドを使用して、相手ピアにデータを送信します。文字列またはバイト配列を送信できます。

```
TsgcRTCPeerConnection *oRTCPeerConnection = new TsgcRTCPeerConnection();  
...  
oRTCPeerConnection->WriteData("Hello from sgcWebSockets!!!");
```

データの受信

TsgcRTCPeerConnection が相手ピアからデータを受信するたびに、OnRTCMessages イベントが呼び出されます。

```
void OnRTCMessages(TObject *Sender, const TBytes aBytes)  
{  
    ShowMessage(TEncoding->UTF8->GetString(aBytes));  
}
```

Datasnap

デフォルトでは、DataSnap は HTTP リクエストを処理するサーバーとして **TIdHttpWebBrokerBridge** を使用します。ただし、このサーバーを、より多くのプロトコルをサポートしてはるかに優れたパフォーマンスを提供し、サーバーパフォーマンスを向上させる HTTP/2 や多数の接続を処理できる IOCP などの最新プロトコルを活用できる別のサーバーに置き換えることで、DataSnap アプリケーションの可能性を広げることができます。

サーバー

サーバー	主な機能	説明
TsgcWSHTTPWebBrokerBridgeServer	WebSocket Protocol HTTP 1.* Protocol XHR Protocol IOCP	Indy ライブラリをベースにしており、同じポートで WebSocket と HTTP プロトコルをサポートします。IOCP も有効にできます。
TsgcWSHTTP2WebBrokerBridgeServer	WebSocket プロトコル HTTP 1.* プロトコル HTTP/2 プロトコル XHR プロトコル IOCP	Indy ライブラリに基づき、同じポートで WebSocket と HTTP/2 プロトコルをサポートします。IOCP も有効にできます。
TsgcWSServer_HTTPAPI_WebBrokerBridge	WebSocket プロトコル HTTP 1.* プロトコル HTTP/2 プロトコル XHR プロトコル IOCP	Microsoft HTTP API の HTTP.SYS に基づき、同じポートで WebSocket と HTTP/2 プロトコルをサポートします。デフォルトで IOCP が使用されます。最高のパフォーマンスのために推奨されます。

TsgcWSHTTPWebBrokerBridgeServer

TsgcWSHTTPWebBrokerBridgeServer は **TIdHttpWebBrokerBridge** をサーバーベースとして使用し、DataSnap、HTTP、WebSocket 接続に単一のサーバーを使用する場合に役立ちます。

TsgcWSHTTPWebBrokerBridgeServer は **TsgcWebSocketHTTPServer** から継承されるため、このサーバーを参照できます。

TIdHttpWebBrokerBridge を TsgcWSHTTPWebBrokerBridgeServer に置き換えるには次の手順に従ってください:

1. TsgcWSHTTPWebBrokerBridgeServer の新しいインスタンスを作成してください。
2. TIdHttpWebBrokerBridge へのすべての呼び出しを TsgcWSHTTPWebBrokerBridgeServer に置き換えます。
3. WebSocket 接続を処理するには [TsgcWebSocketHTTPServer](#) を参照してください。

設定

Datasnap コンポーネントはソースフォルダーにのみ配置されています。これらのオブジェクトは sgcWebSockets パッケージに含まれていないため、コンパイル済みフォルダーには見つかりません。実行時に作成する必要があります。

必要なファイルをプロジェクトに追加するか、sgcWebSockets パッケージの Source フォルダーへのパスを設定するだけです。必要なファイル:

- sgcWebSocket_Server_WebBrokerBridge

プロジェクトが IdHTTPWebBrokerBridge を使用している場合は sgcidHTTPWebBrokerBridge に変更します (Enterprise Edition のみ適用)。

イベント

```
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";
    }
}
```

ロードバランサー

サーバーが [TsgcWebSocketLoadBalancerServer](#) の背後にある場合、CORS の問題が発生する可能性があります。これらの問題を回避するには、次のコードを使用してください。

```
void __fastcall TWebModule1::WebModuleBeforeDispatch(TObject *Sender, TWebRequest *Request, TWebResponse *Response)
{
    Response->SetCustomHeader("Access-Control-Allow-Origin", "*");
    if (Trim(Request->GetFieldByName("Access-Control-Request-Headers")) != "")
    {
        Response->SetCustomHeader("Access-Control-Allow-Headers", Request->GetFieldByName("Access-Control-Request-Headers"));
        Response->Handled = true;
    }
    if (FServerFunctionInvokerAction != nullptr)
    {
        FServerFunctionInvokerAction->Enabled = AllowServerFunctionInvoker;
    }
}
```

TsgcWSHTTP2WebBrokerBridgeServer

TsgcWSHTTP2WebBrokerBridgeServer は、サーバーベースとして **HTTP/2** プロトコルを有効にした **TsgcWebSocketHTTPServer** を使用します。DataSnap、HTTP/2、WebSocket接続に単一のサーバーを使用したい場合に便利です。

TsgcWSHTTP2WebBrokerBridgeServer は **TsgcWebSocketHTTPServer** を継承しているため、このサーバーを参照できます。

TIdHttpWebBrokerBridge を TsgcWSHTTP2WebBrokerBridgeServer に置き換えるには、以下のステップに従ってください。

1. TsgcWSHTTP2WebBrokerBridgeServer の新しいインスタンスを作成します。
2. TIdHttpWebBrokerBridge へのすべての呼び出しを TsgcWSHTTP2WebBrokerBridgeServer に置き換えます。
3. WebSocket 接続を処理するには [TsgcWebSocketHTTPServer](#) を参照してください。

設定

Datasnap コンポーネントはソースフォルダーにのみ配置されています。これらのオブジェクトは sgcWebSockets パッケージに含まれていないため、コンパイル済みフォルダーには見つかりません。実行時に作成する必要があります。

必要なファイルをプロジェクトに追加するか、sgcWebSocketsパッケージのSourceフォルダーへのパスを設定するだけです。必要なファイル:

- sgcWebSocket_Server_Base_WebBrokerBridge
- sgcWebSocket_Server_WebBrokerBridge_HTTP2

プロジェクトが IdHTTPWebBrokerBridge を使用している場合は sgclIdHTTPWebBrokerBridge に変更します (Enterprise Edition のみ適用)。

イベント

```
FServer = new TsgcWSHTTP2WebBrokerBridgeServer();
FServer->OnCommandRequest = OnCommandRequestEvent;
FServer->OnCommandGet = OnCommandGetEvent;

void OnCommandRequestEvent(TIdContext *AThread, TIdHTTPRequestInfo *ARequestInfo,
TIdHTTPResponseInfo *AResponseInfo, ref bool aHandled)
{
    if (ARequestInfo->Document == "/test.html")
    {
        aHandled = true;
    }
}

void OnCommandGetEvent(TIdContext *AThread, TIdHTTPRequestInfo *ARequestInfo, TIdHTTPResponseInfo
*AResponseInfo)
{
    if (ARequestInfo->Document == "/test.html")
    {
        AResponseInfo->ResponseNo = 200;
        AResponseInfo->ContentText = "hello all";
    }
}
```

TsgcWSServer_HTTPAPI_WebBrokerBridge

TsgcWSServer_HTTPAPI_WebBrokerBridge は、サーバーベースとして HTTP/2 プロトコルを有効にした **TsgcWebSocketServer_HTTPAPI** を使用します。DataSnap、HTTP/2、WebSocket接続に単一のサーバーを使用したい場合に便利です。

TsgcWSServer_HTTPAPI_WebBrokerBridge は **TsgcWebSocketServer_HTTPAPI** を継承しているため、このサーバーを参照できます。

TIdHttpWebBrokerBridge を TsgcWSServer_HTTPAPI_WebBrokerBridge に置き換えるには次の手順に従ってください:

1. TsgcWSServer_HTTPAPI_WebBrokerBridge の新しいインスタンスを作成します。
2. TIdHttpWebBrokerBridge へのすべての呼び出しを TsgcWSServer_HTTPAPI_WebBrokerBridge に置き換えます。
3. WebSocket 接続を処理するには [TsgcWSServer_HTTPAPI_WebBrokerBridge](#) を参照してください。

設定

Datasnap コンポーネントはソースフォルダーにのみ配置されています。これらのオブジェクトは sgcWebSockets パッケージに含まれていないため、コンパイル済みフォルダーには見つかりません。実行時に作成する必要があります。

必要なファイルをプロジェクトに追加するか、sgcWebSocketsパッケージのSourceフォルダーへのパスを設定するだけです。必要なファイル:

- sgcWebSocket_Server_Base_WebBrokerBridge
- sgcWebSocket_Server_HTTPAPI_WebBrokerBridge

プロジェクトが IdHTTPWebBrokerBridge を使用している場合は sgclIdHTTPWebBrokerBridge に変更します (Enterprise Edition のみ適用)。

イベント

```
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

OpenAPI仕様は、以前は **Swagger仕様**として知られており、RESTful Webサービスの記述、生成、利用、可視化のための機械可読インターフェイスファイルの仕様です。以前はSwaggerフレームワークの一部でしたが、2016年に独立したプロジェクトとなり、Linux Foundationのオープンソース共同プロジェクトであるOpenAPI Initiativeによって監督されています。Swaggerおよびその他のいくつかのツールは、インターフェイスファイルに基づいてコード、ドキュメント、テストケースを生成できます。

OpenAPIインターフェイスファイルに基づいて実装されたアプリケーションは、メソッド、パラメータ、モデルのドキュメントを自動的に生成できます。これにより、ドキュメント、クライアントライブラリ、ソースコードの同期を維持するのに役立ちます。

Pascal パーサー

sgcOpenAPI ジェネレーター は OpenAPI スペックが与えられると自動的に API クライアントライブラリ（SDK 生成）の生成を可能にします。以下の OpenAPI 仕様がサポートされています：

- **OpenAPI 3.***
- **Swagger 2.*** (2.0から3.0に自動変換)
- **Swagger 1.*** (1.0から3.0に自動変換)

sgcOpenAPIを使用すると、JSON/YAMLのOpenAPIまたはSwaggerから、**Native Pascal Language** でクライアントAPIインターフェイスを **自動的に** 生成できます。現在、Delphi 7から最新のDelphiバージョンまでをサポートしています。

sgcOpenAPI Generator を使用すると、OpenAPI / Swagger 仕様からドキュメントファイルを作成できます。

[OpenAPI Parser Pascal](#) の詳細を読む。

OpenAPI クライアント

生成されたクライアントインターフェイスには、OpenAPI 仕様で定義されたすべての関数/メソッドが含まれます。定数と列挙型も作成されます。

次の**認証**メソッドがサポートされています：

- 基本認証
- [OAuth2 コード](#)（インタラクティブ）
- [OAuth2 認証情報](#)（非インタラクティブ）
- [JWT](#)

[OpenAPI クライアント](#) の詳細を読む。

OpenAPI サーバー

Delphi の組み込み OpenAPI サーバーを使用して OpenAPI 3.0 サービスをホストすることもできます。サーバーはリクエストを仕様に対して自動ルーティングし、/openapi.json で仕様を、/docs で Swagger UI を提供し、CORS およびリクエスト/ボディ検証をすぐに使用できる状態でサポートします。

API を定義する 2 つの方法：

- **仕様優先** — 既存の OpenAPI 3.0 JSON/YAML ファイルをランタイムでロードします。
- **コードファースト** — Delphi クラスに RTTI 属性を付加すると、仕様が自動的に生成されます（Delphi XE7 以降が必要）。

[OpenAPIサーバー](#) の詳細を読む。

API

以下の API がコンパイルされサポートされています：

- [Amazon AWS](#)
- [Google Cloud APIs](#)
- [Microsoft Azure](#)
- [その他のAPI](#)

OpenAPI | Parser Pascal

sgcOpenAPIパーサー は、JSON形式のOpenAPI 3.0仕様を読み取り、Native Pascal CodeでのDelphiクライアントを自動的に作成します。

sgcOpenAPI パーサーは以下の仕様と互換性があります:

- OpenAPI 3.*
- Swagger 2.* (2.0から3.0に自動変換)
- Swagger 1.* (1.0から3.0に自動変換)

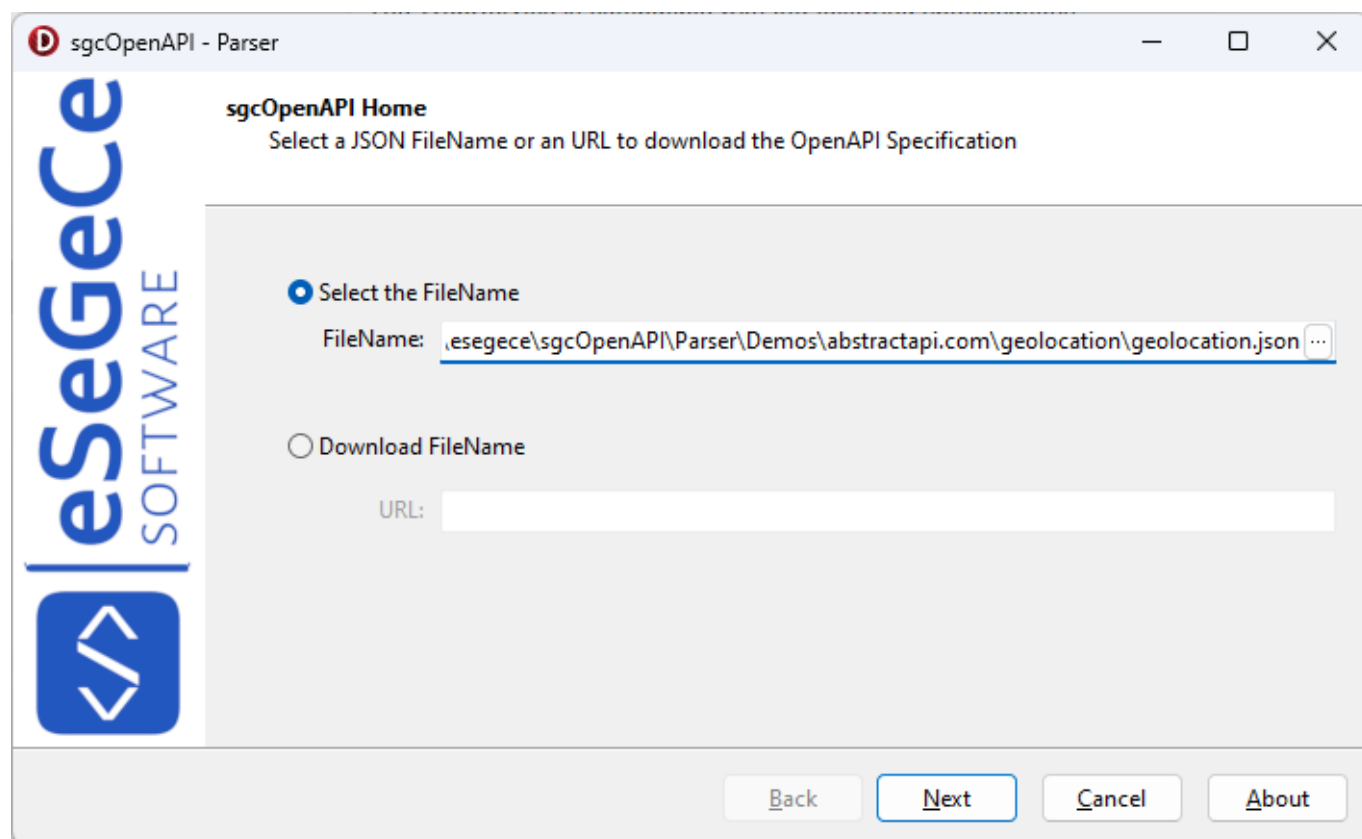
仕様ファイルはJSON形式である必要があります。YAML形式の場合は自動的に変換されます。パーサーは複数のスキーマに分割された仕様のインポートを許可します。URLを指定すると仕様のダウンロードを試み、ファイル名を指定するとローカルドライブから外部スキーマの読み込みを試みます。

OpenAPI パーサー実行ファイルはコマンドラインパラメータを処理するか、ライブラリとして使用できます。

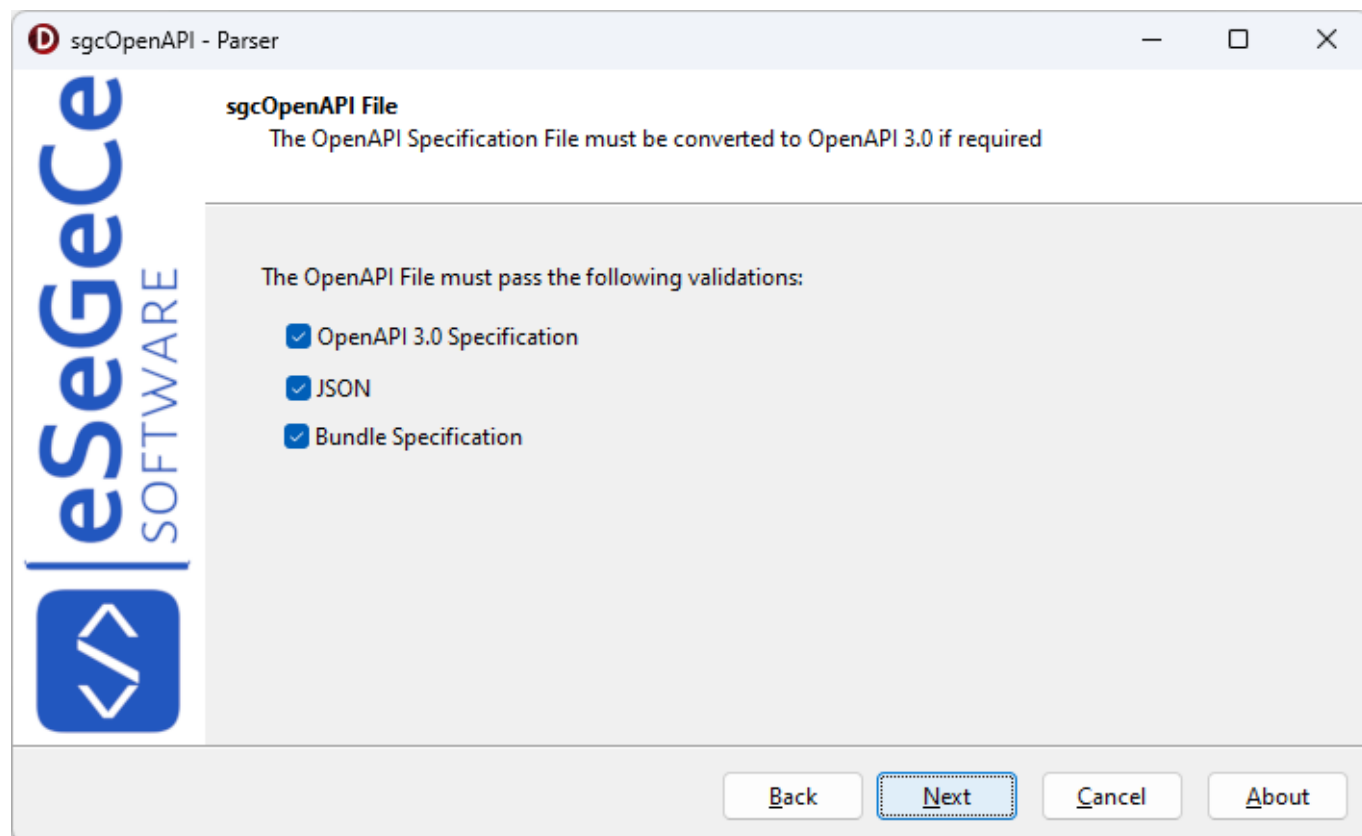
OpenAPI 仕様のインポート

最初のステップは、openAPI仕様をインポートすることです。JSON形式のopenAPI 3.0仕様を入手したら、当社のOpenAPI WebServiceを使用して必要なDelphiファイルを生成できます。以下の手順に従ってください。

- sgcOpenAPI Parserを実行し、インポートする仕様またはダウンロードするURLを選択します。



- 仕様が検証され、パーサーと互換性がある場合は次のステップに進めます。仕様が swagger 2.0 などの古いバージョンを使用している場合は、自動的に変換されます。仕様が複数のスキーマに分割されている場合、パーサーは単一の仕様にバンドルしようとします。ファイルの変換中に問題が発生した場合は、エラーメッセージが表示されます。



- ドキュメントを解析する前に、次のオプションをカスタマイズできます。
 - **ClassName:** メインのクラス名です。デフォルトでは "sgcOpenAPI_" で始まり、仕様ファイル名が追加されます。
 - **Namespace:** 生成される Pascal ファイルの名前です。デフォルトでは拡張子 ".pas" を付けた仕様ファイルと同じ名前になります。
 - **Create Classes:** チェックを入れると、仕様ファイルからクラスが作成されます。例: リクエストで JSON オブジェクトを送信する必要があり、この JSON オブジェクトが仕様に記載されている場合、パーサーは JSON オブジェクトのフィールドを持つクラスを作成します。
 - **Enable Classes:** チェックを入れると、JSON オブジェクトで使用するために仕様ファイルから生成されたクラスが有効になります (Rad Studio XE7 以降が必要)。
 - **Documentation:** チェックを入れると、パーサーはフィールド、クラス、メソッドにコメントを追加します。
 - **Authentication:** 認証が存在する場合はいずれかを選択します。
 - **None:** API は認証メソッドを使用しません。
 - **Basic:** API は BASIC 認証を使用します。
 - **Token:** HTTP ヘッダーとしてトークンを送信する必要があります。このトークンは他の外部メソッドから取得します。
 - **OAuth2:** リクエストは HTTP リクエストの認証に OAuth2 を使用します。
 - **JWT:** リクエストは HTTP リクエストの認証に JWT を使用します。

sgcOpenAPI Parser Options

Customize the options of the OpenAPI parser.

Classname:

Namespace:

Options: ☒ Create Classes ☒ Enable Classes ☒ Documentation

Authentication: ☒ None ☐ Basic ☐ Token ☐ OAuth2 ☐ JWT

Back Next Cancel About

- 解析されたメソッドの一覧を含むグリッドが情報として表示されます。デフォルトでは、パーサーは OperationId をメソッド名として使用しますが、OperationId が定義されていない場合や値が無効な場合は、サマリーをメソッド名として使用するように変更できます。

sgcOpenAPI Parsed Methods

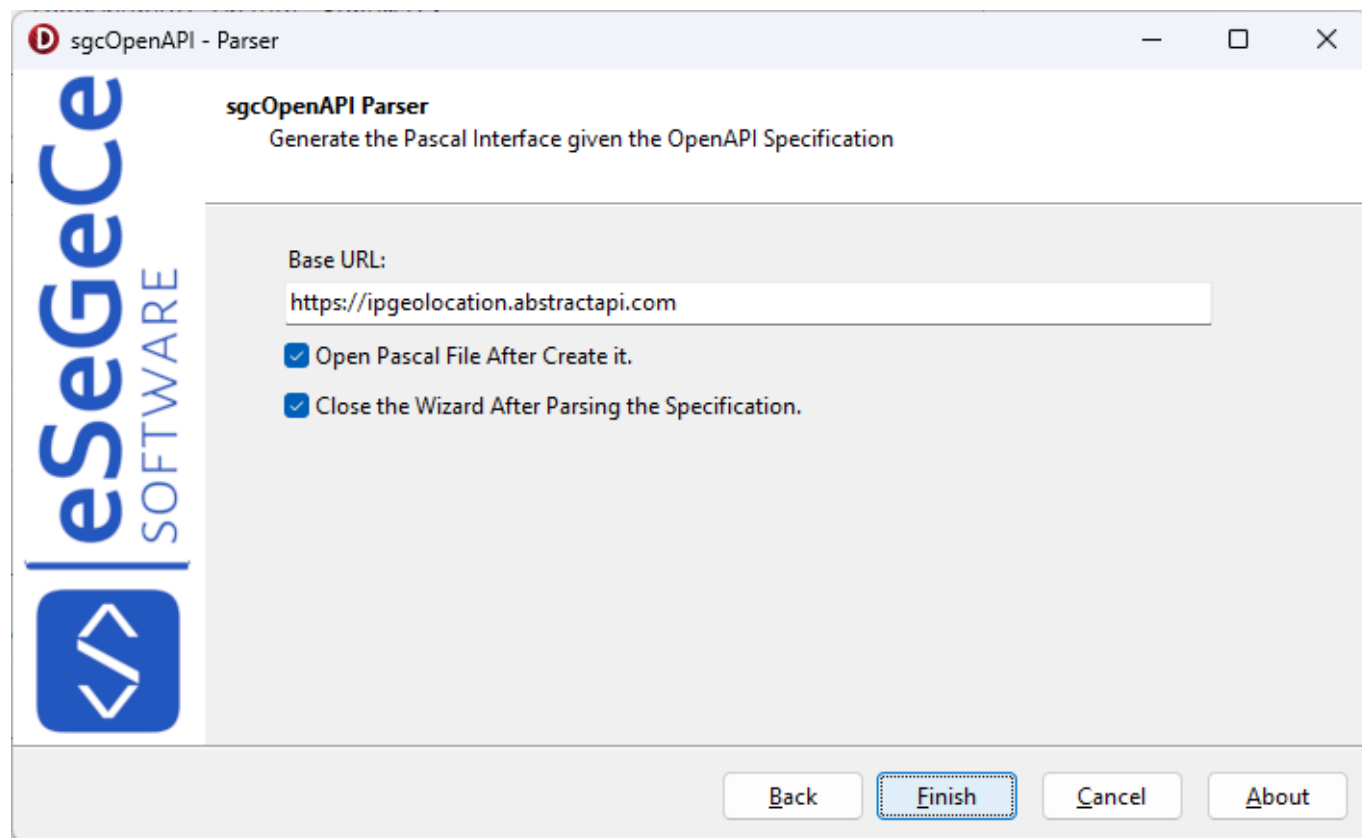
Find below the list of parsed methods.

Import	Method	Description
☒	Retrieve_the_location_of_an_IP_address	Retrieve the location of an IP address

☐ Method Name From Summary

Back Next Cancel About

- 最後に、**Base URL** が正しい値を持っているか確認し、生成されたファイルが作成後に自動的に開かれるかどうか、ファイル生成後にウィザードが閉じるかどうかをカスタマイズできます。



- **Finish**を押して仕様を解析し、pascalファイルを生成します。

使用例

abstractapi.com が IP アドレスの位置を取得するために使用するシンプルな OpenAPI 仕様を使用します。

デモをテストする前に、abstractapi.comで無料アカウントを作成してAPIキーを取得する必要があります。

<https://app.abstractapi.com/users/signup>

sgcOpenAPI Parser Setup を イン ス ト ー ル し、sgcOpenAPI.exe を 開 い て、"Demos\abstractapi.com\geolocation"フォルダにあるOpenAPI仕様をインポートします。インポートして、デモと同じフォルダに"geolocation.pas"という名前で保存したら、デモプロジェクトを開き、コンパイルして実行します。Abstractapiのウェブサイトから取得したAPIキーを入力し、Geolocationを押します。

 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\u0082\u2070",
    "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

ディクショナリ（マップ、ハッシュマップ、または連想配列とも呼ばれる）はキーと値のペアのセットです。OpenAPIではキーが文字列のディクショナリを定義できます。ディクショナリを定義するには、type: objectを使用し、additionalPropertiesキーワードを使用してキーと値のペアの値の型を指定します。例えば、文字列から文字列へのディクショナリ:

```
{
  "en": "English",
  "fr": "French"
}
```

OpenAPI Parser は Embarcadero のJSONクラスを使用しており、TDictionaryをJSONに変換すると、キー/値ペアを作成する代わりに、TDictionaryのすべての内部オブジェクトを作成するため、出力が正しくありません。json文字列をTDictionaryオブジェクトに変換しようとする場合も同様です。

Convert AdditionalProperties to JSON

OpenAPI パーサーは AdditionalProperties クラスを TsgcAdditionalProperties 型として作成します。クラスをJSON 文字列に変換する必要がある場合は、TsgcAdditionalProperties を継承する新しいクラスを作成し、必要なフィールドをすべて作成してからプロパティクラスに割り当てる必要があります。

使用例: 2 つのプロパティを持つクラスで、1 つが追加プロパティの場合、JSON 出力は次のようになります

```
{
  "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;
```

TsgcAdditionalProperties を継承する新しいクラスを作成し、2 つのプロパティを作成します。

```
TCustomAdditionalProperties = class(TsgcAdditionalProperties)
private
  FKey1: string;
  FKey2: string;
public
  property Key1: string read FKey1 write FKey1;
  property Key2: string read FKey2 write FKey2;
end;
```

最後に、このクラスを AdditionalProperties プロパティに割り当てます:

```
TMyClass *oClass = new TMyClass();
oClass->Property1 = "value1";
TCustomAdditionalProperties *oAdditionalProperties = new TCustomAdditionalProperties();
oAdditionalProperties->Key1 = "value1";
oAdditionalProperties->Key2 = "value2";
oClass->AdditionalProperties = oAdditionalProperties;
```

JSON を AdditionalProperties に変換する

JSON クラスの制限により、OpenAPI ジェネレーターは JSON 文字列が解析された後に `TsgcAdditionalProperties.Dictionary` プロパティに `AdditionalProperties` を読み込む追加メソッドを作成します。そのため、`AdditionalProperties` のコンテンツにアクセスするには、`TsgcAdditionalProperties.Dictionary` プロパティにアクセスするだけです。

OpenAPI | コマンドライン

sgcOpenAPI パーサーには、OpenAPI 仕様を Delphi (Pascal) クライアントファイルに変換するコマンドラインインターフェイスが含まれています。パーサーは出力を生成する前に仕様 (OpenAPI 3 JSON + バンドルされたスキーマ) を検証して正規化します。

使用法

基本構文

```
sgcOpenAPI.exe -i "c:\openapi.json" -o "c:\openapi.pas"
```

パラメータ

すべてのパラメータは大文字と小文字を区別しません。短い形式と長い形式は同等です。

- **-i -input** : [必須] OpenAPI 仕様へのパスまたはダウンロード用 URL。YAML または Swagger 1/2 の場合は自動的に OpenAPI 3 JSON に変換されます。
- **-o -output** : [必須] 生成する出力 Pascal ファイル (Delphi クライアント)。
- **-nc -nclasses** : [オプション] スキーマの Delphi クラスを生成しない (インターフェース/クライアントメソッドのみ)。
- **-nj -njson** : [オプション] 生成されたクラスの JSON シリアライズを有効にしません。
- **-nd -ndoc** : [オプション] 出力にドキュメントコメントを生成しません。
- **-m -method** : [オプション] 生成されたメソッドのメソッド命名戦略:
 - **0** : OperationId を使用 (デフォルト)。
 - **1** : Summary を使用。
 - **2** : Endpoint パスを使用。
- **-a -auth** : [オプション] 生成されたクライアントで有効にする認証モード:
 - **0** : なし。
 - **1** : Basic。
 - **2** : トークン。
 - **3** : OAuth2。
 - **4** : JWT。
- **-u -url** : [オプション] OpenAPI ファイルで検出されたベース URL を上書きします。
- **-l -log** : [オプション] 解析中にログ出力を表示します。
- **-user** : [オプション] ライセンスユーザー名 (ライセンスが有効化されていない場合のみ必要)。
- **-password** : [オプション] ライセンスパスワード (ライセンスがアクティベートされていない場合にのみ必要)。
- **-h -help** : コマンドラインヘルプを表示します。

使用例

1) ローカルファイルから Delphi クライアントを生成する

```
sgcOpenAPI.exe -i "c:\specs\openapi.json" -o "c:\clients\orders.pas"
```

2) URL からクライアントを生成し、クラスとドキュメントを無効化する

```
sgcOpenAPI.exe -i "https://api.example.com/openapi.json" -o "c:\clients\api.pas" -nc -nd
```

3) サマリーをメソッド名として使用し、OAuth2 を有効にする

```
sgcOpenAPI.exe -i "c:\specs\openapi.json" -o "c:\clients\auth.pas" -m 1 -a 3
```

4) ベース URL を上書きしてログ出力を表示する

```
sgcOpenAPI.exe -i "c:\specs\openapi.json" -o "c:\clients\override.pas" -u "https://api.example.com/v2" -l
```

注意

- 入力仕様が有効でないか、バンドルされた OpenAPI 3 JSON ファイルでない場合、パーサーは生成前に自動的に変換してバンドルを試みます。
- URL を使用する場合、パーサーは検証と出力生成前に仕様をダウンロードします。

OpenAPI | ライブラリ

OpenAPI パーサーは共有ライブラリ（DLL）として使用できるため、コマンドラインツールを呼び出すことなく独自のアプリケーションから Pascal クライアントを生成できます。ライブラリは CLI で利用可能な同じパーサー機能をラップする少数の関数をエクスポートします。

Library functions

- **CreateOpenAPIPascalFile** - デフォルトオプションを使用して OpenAPI 仕様から Pascal クライアントファイルを作成します。
- **CreateOpenAPIPascalFileEx** - パーサーオプションを完全に制御して Pascal クライアントファイルを作成します。
- **GetOpenAPILastErrors** - パーサーが生成した最後のエラーメッセージを返します。

CreateOpenAPIPascalFile

デフォルトのパーサーオプション（クラスの生成、JSON サポート、ドキュメント、**OperationId** からのメソッド名、認証なし、ベース URL のオーバーライドなし）を使用して Pascal クライアントファイルを作成します。

```
typedef bool (__stdcall *TCreateOpenAPIPascalFile)(const char* aInputFile, const char* aOutputFile);
```

パラメータ

- **aInputFile**: 解析する OpenAPI 仕様のフルパス (または URL)。ファイルが YAML または Swagger 1/2 の場合、パーサーは出力生成前に OpenAPI 3 JSON に正規化します。
- **aOutputFile**: 作成する Pascal ユニットの完全パス。

戻り値

ファイルが正常に生成された場合は **True** を、そうでない場合は **False** を返します。失敗した場合は、**GetOpenAPILastErrors** を呼び出してエラーメッセージを取得してください。

使用例

CreateOpenAPIPascalFileEx

パーサーオプションを明示的に制御して Pascal クライアントファイルを作成します。

```
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);
```

パラメータ

- **aInputFile**: 解析する OpenAPI 仕様のフルパス (または URL)。
- **aOutputFile**: 作成する Pascal ユニットの完全パス。
- **aGenerateClasses**: **True** の場合、ジェネレーターはスキーマで定義された Pascal モデル（クラス）を作成します。**False** の場合、クライアントインターフェースのみが生成されます。
- **aGenerateJSON**: **True** の場合、クラスに JSON シリアライズヘルパーが生成されます。**False** の場合、JSON サポートは省略されます。
- **aGenerateDocumentation**: **True** の場合、OpenAPI 仕様のドキュメントファイルが生成されます。

- **aMethodName:** メソッド名の生成方法を制御します:
 - **0** = OperationId (デフォルト)。
 - **1** = Summary。
 - **2** = Endpoint。
- **aAuthentication:** クライアントに認証ヘルパーを追加します:
 - **0** = なし。
 - **1** = ベーシック。
 - **2** = トークン。
 - **3** = OAuth2。
 - **4** = JWT。
- **aBaseURL:** OpenAPI 仕様のベース URL を上書きします。元の値を保持するには空文字列を渡してください。
- **aEnableLog: True** の場合、パーサーはトラブルシューティングに役立つ進行状況と検証メッセージを書き出します。

戻り値

ファイルが正常に生成された場合は **True** を、そうでない場合は **False** を返します。失敗した場合は、**GetOpenAPILastErrors** を呼び出してエラーメッセージを取得してください。

使用例

GetOpenAPILastErrors

パーサーが生成した最後のエラー（検証、変換、または解析エラー）を説明するメッセージを返します。**CreateOpenAPIPascalFile** または **CreateOpenAPIPascalFileEx** が **False** を返した後にこの関数を呼び出してください。

```
typedef const char* (__stdcall *TGetOpenAPILastErrors)();
```

戻り値

エラーの数を返します

使用例

OpenAPI | API

sgcOpenAPI パーサーは、解析された Pascal ファイルのより詳細な情報を取得するためにカスタム dll を使用できます。**sgcOpenAPI_CreateSection** メソッドを持つ新しい dll を作成し、パーサーと同じフォルダにライブラリをコピーするだけです。新しいファイルセクションが作成されるたびに API の CreateSection メソッドが呼び出され、必要に応じてコンテンツを変更できます。以下に Delphi の簡単な例を示します：

```
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 | クライアント

TsgcOpenAPI_Client は OpenAPI 仕様から HTTP リクエストを行うための主要なメソッドとプロパティをカプセル化する非ビジュアルコンポーネントです。

sgcOpenAPI Parserで作成されるすべてのOpenAPIインターフェイスには**2個のメソッド**があります

1. **GetOpenAPIClient** : メインクラスのインスタンスを返すシングルトン関数で、存在しない場合は自動的に作成します。
2. **FreeOpenAPIClient**: 作成されている場合はメインクラスを解放します。

使用例

Abstractapi を使用して IP アドレスのローカライズ情報を取得します。

```
GetOpenAPIClient->Retrieve_the_location_of_an_IP_address("your api", "80.258.15.2");
```

認証

- **Basic**: 認証にHTTPヘッダーとしてユーザー名とパスワードを使用します。
 - **UserName**: ユーザー名。
 - **Password**: シークレット。
- **OAuth2**: OAuth2 を使用して認証します。2 種類の認可をサポートします:
 - **auth2Code**: シングルページアプリケーション、ウェブアプリケーション、ネイティブインストールアプリケーションを含む大多数のアプリケーションタイプで認証と認可を行うために使用されます。このフローにより、アプリはリソースへのアクセスに使用できる access_token、追加の access_token を取得するためのリフレッシュトークン、およびサインインユーザーの ID トークンを安全に取得できます。
 - **auth2ClientCredentials**: このタイプのグラントは、ユーザーとの即時インタラクションなしにバックグラウンドで実行する必要があるサーバー間インタラクションに一般的に使用されます。このタイプのアプリケーションはデーモンまたはサービスアカウントと呼ばれることが多いです。
 - 詳しくは [OAuth2](#) を参照してください。
- **JWT**: JWT を使用して認証します。 [JWT](#) の詳細をご覧ください。
- **Token**: Bearer トークンまたは任意のカスタムトークン値を渡すことができます。

TLSOptions

HTTP/1 プロトコルを使用してセキュアな SSL/TLS サーバーへの接続方法を設定できます

ALPNProtocols: サーバーに送信される ALPN プロトコルのリスト。

RootCertFile: ルート証明書ファイルへのパス。

CertFile: 証明書ファイルへのパス。

KeyFile: 証明書鍵ファイルへのパス。

Password: 証明書がパスワードで保護されている場合は、ここに設定します。

VerifyCertificate: 証明書を検証する必要がある場合は、このプロパティを有効にしてください。

VerifyDepth: X.509 証明書の検証実行時に許可されるリンクの最大数を表す整数プロパティ。

Version: デフォルトでは TLS 1.0 を使用します。サーバーがより高い TLS バージョンを要求する場合、ここで選択できます。

IOHandler: TLS を使用した接続に使用するライブラリを選択します。

iohOpenSSL: OpenSSLライブラリを使用し、Indyコンポーネントのデフォルトです。win32/win64用のopensslライブラリをデプロイする必要があります。

iohSChannel: Microsoft が Windows 用に実装したセキュリティプロトコルである Secure Channel を使用します。OpenSSL ライブラリをデプロイする必要がありません。Windows 32/64 ビットでのみ動作します。

OpenSSL_Options: OpenSSL ライブラリの設定。

APIVersion: 使用する OpenSSL API を定義できます。

oslAPI_1_0: API 1.0 OpenSSL を使用します。Indy で最後にサポートされたバージョンです。

oslAPI_1_1: API 1.1のOpenSSLを使用します。カスタムIndyライブラリが必要で、OpenSSL 1.1.1ライブラリ（TLS 1.3サポート付き）の使用が可能になります。

oslAPI_3_0: API 3.0 OpenSSLを使用します。当社のカスタムIndyライブラリが必要であり、OpenSSL 3.0.0ライブラリの使用を可能にします(TLS 1.3サポート付き)。

LibPath: opensslライブラリの場所を設定できます

oslpNone: これはデフォルトです。opensslライブラリは、バイナリと同じフォルダー、または既知のパスにある必要があります。

oslpDefaultFolder: すべてのIDEパーソナリティに対してライブラリが配置されるべきopensslパスを自動的に設定します。

oslpCustomFolder: このオプションが選択されている場合、プロパティ LibPathCustom にフルパスを定義します。

LibPathCustom: LibPath = oslpCustomFolder の場合、openssl ライブラリが格納されているフォルダーの完全パスをここで定義します。

UnixSymLinks: Unix システムでシンボリックリンクの読み込みを有効または無効にします（デフォルトでは有効です。OSX64 は除きます）。

oslsSymLinksDefault: デフォルトでは有効です。ただし OSX64 では無効です（MacOS Monterey 以降、バージョンなしでライブラリを読み込もうとすると失敗するため）。

oslsSymLinksLoadFirst: SymLinksを読み込み、バージョンライブラリを読み込む前に行います。

oslsSymLinksLoad: バージョンライブラリの読み込みを試みた後にシンボリックリンクを読み込みます。

oslsSymLinksDontLoad: SymLinksを読み込みません。

SChannel_Options: Windows 証明書ストアから証明書を使用できます。

CertHash: 証明書のハッシュ値です。PowerShell で dir コマンドを実行することで証明書のハッシュ値を確認できます。

CipherList: 使用する暗号スイートを(";"で区切って)設定できます。例: CALG_AES_256;CALG_AES_128

CertStoreName: 証明書が保存されているストアの名前。以下のいずれかを選択します:

scsnMY (デフォルト)

scsnCA

scsnRoot

scsnTrust

CertStorePath: 証明書が格納されているストアパス。以下から選択してください:

scspStoreCurrentUser (デフォルト)

scspStoreLocalMachine

プロキシオプション

このプロパティを使用して、プロキシ経由の接続を設定します。

Enabled: プロキシ接続を有効にするには true に設定します。

Host: プロキシサーバーアドレス。

Port: プロキシサーバーのポート

UserName/Password: プロキシへの接続認証。必要な場合のみ。

ProxyType: 以下のプロキシがサポートされています:

- HTTP
- Socks4
- Socks4A
- Socks5

ログ

Log プロパティが有効な場合、ソケットメッセージを指定されたログファイルに保存します。デバッグに役立ちます。

Log: HTTPリクエストをテキストファイルに保存する場合は有効にします。

LogFileName: ファイル名への完全パス。

プロパティ

OpenAPI クライアントのカスタマイズに使用できるその他のプロパティ：

EncodeBodyAsUTF8: 有効にすると、JSON ボディが UTF8 でエンコードされます（デフォルトは false）。

イベント

OpenAPIクライアントを使用する際に処理できるイベントの一覧を以下に示します。

OnBeforeRequest

このイベントは、HTTPリクエストが呼び出される前に呼び出されます。パラメータ名、ヘッダー、セキュリティなどをカスタマイズできます。以下に、一部のパラメータの名前を置き換える方法の例を示します。

```
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

このイベントはファイルがアップロードされたときに呼び出されます。このイベントを使用してアップロードの進捗を確認できます。

OnDownload

このイベントはファイルがダウンロードされたときに呼び出されます。このイベントを使用してダウンロードの進捗を確認できます。

OnSSLVerifyPeer

証明書の検証が有効な場合、このイベントでサーバー証明書を検証して受け入れるかどうかを決定できます。

OnSSLGetHandler

このイベントはSSLハンドラーが作成される前に発生します。ここで独自のSSLハンドラーを作成し（TIdServerIOHandlerSSLBaseまたはTIdIOHandlerSSLBaseから継承する必要があります）、必要なプロパティを設定できます

OnSSLAfterCreateHandler

カスタムSSLオブジェクトが作成されていない場合、デフォルトでOpenSSLハンドラーを使用して作成されます。SSLハンドラーのプロパティにアクセスして、必要に応じて変更できます。

OpenAPI | サーバー

sgcOpenAPI サーバーは、OpenAPI 3.0 仕様によって記述された HTTP サービスをホストします。ルートは仕様の **paths** セクションから自動的に構築され、Swagger UI がすぐに利用でき、受信リクエストは各操作で宣言された JSON スキーマに対して検証できます。サーバーをホストするために 2 つのコンポーネントが提供されています:

- **TsgcHTTPServer_OpenAPI**: スタンドアロン HTTP サーバー。自己完結型で、Indy HTTP サーバーを内蔵しており、sgcWebSockets を必要としません。OpenAPI サービスのみを公開する場合に推奨されます。
- **TsgcWSAPIServer_OpenAPI**: sgcWebSockets と互換性があります。既存の **TsgcWebSocketHTTPServer** にアタッチすることで、同一サーバーが WebSocket と HTTP/OpenAPI の両方のトラフィックを処理できます。

両方のコンポーネントは同じエンジン、オプション、イベントを共有しています。

TsgcHTTPServer_OpenAPI

ユニット **sgcHTTPServer_OpenAPI** で定義されたスタンドアロンサーバー。リッスンポートを設定し、OpenAPI 仕様を読み込んでサーバーをアクティブにしてください。

TsgcWSAPIServer_OpenAPI

sgcWebSockets と互換性があり、ユニット **sgcWebSocket_Server_API_OpenAPI** で定義されています。既存の **TsgcWebSocketHTTPServer** を **Server** プロパティに割り当てて、単一のサーバーインスタンスが WebSocket フレームと OpenAPI HTTP リクエストの両方をディスパッチできるようにします。内部的には、スタンドアロンコンポーネントが使用するのと同じエンジンに委譲するため、API サーフェスは同一です。

組み込みエンドポイント

サーバーは以下のエンドポイントを自動的に提供します (各エンドポイントは **OpenAPIOptions.Endpoint** で無効化できます)。

- /openapi.json**: ロードされた OpenAPI 仕様を返します。 **OpenAPIOptions.Endpoint.ServeSpec** によって制御されます。
- /docs**: API を参照して試すための組み込み Swagger UI です。 **OpenAPIOptions.Endpoint.ServeSwaggerUI** で制御されます。
- OPTIONS ***: CORS プリフライトレスポンス。 **OpenAPIOptions.CORS.Enabled** が True の場合に送信されます。

OpenAPIOptions

TsgcOpenAPIServer_Options 型のトップレベル設定プロパティ。3 つのサブオブジェクトをグループ化します。

エンドポイント: 組み込みエンドポイントとスペックの読み込み。

BasePath: スペックで宣言されたすべてのルートの先頭に付加されるオプションの URL プレフィックス。

SpecFile: 割り当てられている場合、サーバーが **Active** になったときにファイルが自動的に読み込まれます。

ServeSpec: True (デフォルト) の場合、読み込まれた仕様は **/openapi.json** で提供されます。

ServeSwaggerUI: True の場合 (デフォルト)、Swagger UI が **/docs** で提供されます。

Validation: リクエスト検証フラグ。すべてデフォルトは False です。

ValidateRequest: リクエスト全体の検証を有効にするマスタースイッチ。

ValidateRequestBody: 要求本文を、仕様で宣言された JSON-Schema に対して検証します。

ValidateQueryParams: クエリパラメータを宣言されたスキーマに対して検証します。

ValidatePathParams: パスパラメータを宣言されたスキーマに対して検証します。

ValidateRequired: 必須として宣言されたパラメータが欠落している場合にリクエストを拒否します。

CORS: クロスオリジンのサポート。

Enabled: True の場合、すべてのレスポンスに CORS ヘッダーが付加され、**OPTIONS** プリフライトリクエストに自動的に応答します。

AllowOrigins: **Access-Control-Allow-Origin** の値 (デフォルト *)。

AllowHeaders: **Access-Control-Allow-Headers** の値（デフォルト **Content-Type, Authorization**）。

AllowMethods: **Access-Control-Allow-Methods** の値（デフォルト: **GET, POST, PUT, DELETE, PATCH, OPTIONS**）。

イベント

OpenAPI サーバー使用時に処理できるイベントの一覧を以下に示します。

OnBeforeRequest: リクエストが **OnRequest** にディスパッチされる前に発生します。**Accept** パラメータを **False** に設定してリクエストを 403 レスポンスで拒否します。

OnRequest: メインハンドラー。**aOperationId** を検査し、**aContext** を使用してリクエストを読み取り、レスポンスを書き込みます。レスポンスを生成した後、**Handled** を **True** に設定します。

OnAfterRequest: ユーザーハンドラーが返った後に発生します。ログやメトリクスに便利です。

OnAuthenticate: 認証が必要な場合にディスパッチ前に発生します。401 レスポンスで拒否するには **Authenticated** を **False** に設定してください。

OnValidationError: 1つ以上の検証ルールが失敗したときに発火します。**aErrors**を確認し、**Continue**を **False**に設定してリクエストを400レスポンスで拒否します。

OnException: 未処理の例外がユーザーハンドラーから伝播したときに発生します。デフォルトの 500 を変更する必要がある場合は **aResponseCode** を調整します。

Spec-First と Code-First

サーバーは2つの作成アプローチをサポートします:

- **Spec-First:** OpenAPI 3.0 の JSON/YAML ファイルが信頼できる唯一の情報源であり、サーバーはそれを解析してルートを検出します。
- **コードファースト:** RTTI 属性でアノテーションされた Delphi クラスが唯一の情報源です。OpenAPI 仕様は起動時に生成されます。Delphi XE7 以降が必要です。

OpenAPI | サーバー | Spec-First

スペックファーストアプローチでは、OpenAPI 3.0 仕様（JSON または YAML）が信頼できる情報源です。これはすべてのパス、操作、パラメーター、スキーマを記述します。サーバーが起動すると、仕様を解析してルートテーブルを自動的に構築します。各操作のビジネスロジックを提供するだけで済みます。

API コントラクトがすでに存在する場合、コンシューマーと共有されている場合、または Delphi コードベース外で管理されている場合に推奨されるワークフローです。

仕様の読み込み

ディスクから仕様を読み込むには **LoadFromFile** を、インメモリ文字列（例：データベースから取得した仕様や実行時に生成された仕様）から読み込むには **LoadFromString** を使用してください。

ロードされると、ルートテーブルはスペックの **paths** セクションから構築されます。**/pets/{petId}** などのテンプレートセグメントは自動的に認識され、キャプチャされた値は **aContext.PathParamAsXxx** を通じて公開されます。

操作の処理

サーバーはすべての受信リクエストを **OnRequest** イベントにディスパッチし、スペックで宣言された **operationId** を渡します。シンプルな **if/else** チェーン（または辞書ルックアップ）を使用して、操作を適切なハンドラーにルーティングしてください。

リクエストデータの読み取り

TsgcOpenAPIServerContext オブジェクトは、リクエストのすべての部分に対して型付きアクセサを公開します。

レスポンスの書き込み

コンテキストオブジェクトは、レスポンスを書き込むヘルパーも提供します。最も一般的なヘルパーは **RespondJSON** と **RespondError** です。レスポンスを完全に制御するには、**aContext.Response.Code** と **aContext.Response.ContentText** を直接設定してください。

検証

OpenAPIOptions.Validation の検証フラグが有効な場合、サーバーは **OnRequest** を呼び出す前に、仕様で宣言された JSON スキーマに対してすべてのリクエストを検証します：

ValidateRequestBody: リクエストボディは、操作の **requestBody** で参照されるスキーマに対して検証されます。

ValidateQueryParams: クエリ文字列パラメータは宣言されたスキーマに対して検証されます。

ValidatePathParams: パスパラメーターは宣言されたスキーマに対して検証されます。

ValidateRequired: 必須と宣言されたパラメータが欠落している場合、リクエストは拒否されます。

検証が失敗した場合、**OnValidationError** イベントがエラーリストとともに発生します。**Continue** を **False** に設定すると 400 レスポンスでリクエストを拒否します。True のままにすると **OnRequest** ハンドラーが決定します。

デモ

完全なスペックファーストのサンプルが **Demos/30.Server/02.SpecFirst/** に同梱されています。

OpenAPI | サーバー | コードファースト

コードファーストアプローチでは、Delphi クラスが信頼できるソースです。メソッド、パラメーター、RTTI 属性が API コントラクトを記述し、OpenAPI 3.0 仕様は起動時に生成されます。生成後、仕様はファイルから来た場合と同様に **LoadFromString** でサーバーに読み込まれ、ルートテーブルが構築されます。

Code-Firstは、属性を検査するために新しいRTTIに依存するため、**Delphi XE7以降**が必要です。この機能はユニット **sgcHTTP_OpenAPI_Server_CodeFirst** に存在し、コンパイラディレクティブ **{\$IFDEF DXE7}** でガードされています。

サービスの定義

Delphi クラスに属性を付与して API を記述します。メソッド本体はスタブです。実際のロジックはサーバーの **OnRequest** イベントに記述します。メソッドは RTTI スキャナーがシングネチャを読み取るためだけに存在します。

利用可能な属性

スキャナーは以下の属性を認識します。クラスレベルの属性はサービス全体を、メソッドレベルの属性は単一の操作を、パラメータレベルの属性は値の取得元を記述します。

sgcServiceContract(name, description, version): クラスレベル。OpenAPI の **info** ブロックを設定します。

sgcRoute(path): クラスレベルのベースパス、またはクラスベースパスに対するメソッドレベルのルート。

sgcHttpGet, sgcHttpPost, sgcHttpPut, sgcHttpDelete, sgcHttpPatch, sgcHttpHead, sgcHttpOptions: メソッドレベル。オペレーションの HTTP 動詞を宣言します。

sgcSummary(text): メソッドレベル。仕様および Swagger UI に表示される短いサマリー。

sgcDescription(text): メソッドレベル。仕様の長い説明。

sgcTag(name): メソッドレベル。Swagger UI で同じタグの下に操作をグループ化します。

sgcResponse(code, description): メソッドレベル。HTTP ステータスコードと説明を含むレスポンスを宣言します。複数のレスポンスを宣言するために繰り返し使用できます。

sgcFromPath, sgcFromQuery, sgcFromBody, sgcFromHeader: パラメータレベル。パラメータがどこから読み込まれるかを宣言します。

sgcRequired: パラメータレベル。パラメータを必須としてマークします。**ValidateRequired** が有効になっているとき、欠落しているパラメータは検証によって報告されます。

仕様の生成と読み込み

TsgcOpenAPICodeFirstScanner を使用してアノテーションされたクラスから OpenAPI 3.0 仕様を生成し、結果をサーバーに読み込みます。

操作の処理

スキャナーはメソッド名から **operationId** を導出します。メソッド **ListTasks** は操作 **ListTasks** になります。同じ **operationId** を使用して **OnRequest** でディスパッチします。

デモ

完全なコードファーストのサンプルが **Demos/30.Server/01.CodeFirst/** に付属しています。

OpenAPI | Amazon AWS

sgcOpenAPI Amazon AWS クライアント (`TsgcOpenAPI_Amazon_Client`) は、[TsgcOpenAPI_Client](#) を継承する独自の OpenAPI クライアントを持っています。

このコンポーネントには、Amazon AWS Serversに接続するために必要なすべての構成を含む**AmazonOptions** というプロパティがあります。

AmazonOptions

AmazonOptions では、Amazon AWS サーバーに対して認証するために必要な **AccessKey** と **SecretKey** (Amazon アカウントから事前に生成する必要があります) を定義できます。

アクセスキーはリソースへのプログラムのアクセスを付与します。つまり、AWS アカウントのルートユーザーのサインイン認証情報と同様に、アクセスキーを慎重に管理する必要があります。

以下のベストプラクティスを実行することをお勧めします。

1. **IAM ユーザーを作成し、そのユーザーの権限をできるだけ狭く定義**します。
2. そのIAMユーザーの下で**アクセスキーを作成**します。

認証情報を取得したら、次のプロパティに設定します:

- `AmazonOptions.AccessKey`
- `AmazonOptions.SecretKey`

AmazonOptions.JSON プロパティを使用して、レスポンスがJSONかXML形式かを定義できます。

AWS IAM Identity Center (AWS Single Sign-On の後継) の IAM ロール、ユーザー、およびフェデレーションユーザーは一時的なセキュリティ資格情報を持っています。一時的なセキュリティ資格情報は、定義された期間が経過するか、ユーザーがセッションを終了すると期限切れになります。以下のプロパティで一時的な資格情報のトークンを設定できます。

- `AmazonOptions.SessionToken`

最も一般的な使用方法

- **構成**
 - [Amazon AWS Credentials](#)
- **API**
 - [Amazon AWS S3](#)

sgcOpenAPI AWS SDK

現在利用可能な API のリストを以下に示します。

- 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
- MongoDB 互換性を備えた Amazon DocumentDB
- 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 シグナリングチャンネル
- 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 メールサービス
- 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 サービス
- 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 データプレーン
- 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 プロジェクトサービス
- 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 パブリック API
- FinSpace ユーザー環境管理サービス
- ファイアウォール管理サービス
- Kafka のマネージドストリーミング
- Kafka Connect のマネージドストリーミング
- Redshift Data API Service
- Route53 Recovery Cluster
- スキーマ
- Service Quotas
- シンセティクス

OpenAPI Amazon AWS | 認証情報

AWSはアクセス方法に応じて異なる種類のセキュリティ認証情報を必要とします。たとえば、AWS Management Consoleにサインインするにはユーザー名とパスワードが必要で、AWSへのプログラマ的な呼び出しを行うにはアクセスキーが必要です。

考慮事項

- 以下の情報を安全な場所に保存してください。AWSアカウントに関連付けられたメールアドレス、AWSアカウントID、ルートユーザーパスワード、およびアカウントアクセスキーです。ルートユーザーパスワードを忘れたり紛失したりした場合は、リセットするためにアカウントに関連付けられたメールアドレスへのアクセスが必要です。アクセスキーを忘れたり紛失したりした場合は、アカウントにサインインして新しいキーを作成する必要があります。
- 日常のAWSタスクに使用する管理者権限を持つIAMユーザーを作成し、ルートユーザーのパスワードとアクセスキーを安全に保管することを強くお勧めします。ルートユーザーのみが制限されたタスクには、ルートユーザーを使用してください。
- セキュリティ認証情報はアカウント固有です。複数のAWSアカウントへのアクセス権を持っている場合は、アカウントごとに別々の認証情報があります。
- AWS資格情報を第三者に提供しないでください。

プログラムによるアクセス

AWSへのプログラマ的な呼び出しを行うには、AWSアクセスキーを提供する必要があります。

アクセスキーを作成する際に、アクセスキーID (例: AKIAIOSFODNN7EXAMPLE) とシークレットアクセスキー (例: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY) をセットとして作成します。シークレットアクセスキーは作成時のみダウンロード可能です。シークレットアクセスキーをダウンロードしなかった場合や紛失した場合は、新しいものを作成する必要があります。

ユーザーごとに最大2つのアクセスキーを割り当てられます (ルートユーザーまたはIAMユーザー)。2つのアクセスキーを持つことは、ローテーションを行う際に便利です。アクセスキーを無効にすると使用できなくなりますが、2つのアクセスキーの上限にはカウントされます。アクセスキーを削除すると、永久に消去されて復元できませんが、新しいアクセスキーに置き換えることができます。

ルートユーザーとしてサインインしているときにアクセスキーを管理するには

- ルートユーザーとしてAWS管理コンソールにサインインします。
- 右上のナビゲーションバーでアカウント名または番号を選択し、**マイセキュリティ認証情報**を選択してください。
- アクセスキー (アクセスキーIDとシークレットアクセスキー)** セクションを展開します。
- 次のいずれかを行います:
 - アクセスキーを作成するには、**新しいアクセスキーを作成**を選択します。すでに2つのアクセスキーがある場合、このボタンは無効になっており、新しいキーを作成する前にアクセスキーを削除する必要があります。プロンプトが表示されたら、**アクセスキーを表示** または **キーファイルをダウンロード**のいずれかを選択します。これがシークレットアクセスキーを保存する唯一の機会です。シークレットアクセスキーを安全な場所に保存したら、**閉じる**を選択します。
 - アクセスキーを無効にするには、**非アクティブにする**を選択します。確認を求めるプロンプトが表示されたら、**非アクティブ化**を選択します。非アクティブ化されたアクセスキーは、2つのアクセスキーの制限にカウントされます。
 - アクセスキーを有効にするには、**アクティブにする**を選択します。
 - 不要になったアクセスキーを削除するには、アクセスキーIDをコピーしてから**削除**を選択します。アクセスキーを削除する前に、**非アクティブ化**を選択する必要があります。永久に削除する前に、アクセスキーが使用されていないことを確認することをお勧めします。削除を確認するには、テキスト入力フィールドにアクセスキーIDを貼り付けてから**削除**を選択します。

IAMユーザーとしてサインインしているときにアクセスキーを管理するには

- IAMユーザーとしてAWS Management Consoleにサインインします。
- 右上のナビゲーションバーでユーザー名を選択し、**マイセキュリティ認証情報**を選択してください。

3. 次のいずれかを行います：

- アクセスキーを作成するには、**アクセスキーを作成**を選択します。すでに2つのアクセスキーがある場合、このボタンは無効になり、新しいアクセスキーを作成する前に既存のアクセスキーを削除する必要があります。プロンプトが表示されたら、**シークレットアクセスキーを表示** または **.csvファイルをダウンロード**を選択します。これはシークレットアクセスキーを保存する唯一の機会です。安全な場所にシークレットアクセスキーを保存したら、**閉じる**を選択します。
- アクセスキーを無効化するには、**非アクティブにする**を選択します。確認のプロンプトが表示されたら、**非アクティブにする**を選択します。無効化されたアクセスキーは2つのアクセスキーの制限にカウントされます。
- アクセスキーをアクティブにするには、**アクティブにする**を選択します。確認のプロンプトが表示されたら、**アクティブにする**を選択します。
- 不要になったアクセスキーを削除するには、アクセスキーIDをコピーして **削除**を選択します。これによりアクセスキーが無効化されます。完全に削除する前に、アクセスキーが使用されていないことを確認することをお勧めします。削除を確認するには、テキスト入力フィールドにアクセスキーIDを貼り付けて **削除**を選択します。

sgcOpenAPI 設定

独自のAWS Access Keysを取得したら、Amazon AWSサーバーにリクエストを行う前に、OpenAPI Amazon Clientで構成する必要があります。

```
GetOpenAPIClient->AmazonOptions->AccessKey = "AKIAIOSFODNN7EXAMPLE";
GetOpenAPIClient->AmazonOptions->SecretKey = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY";
```

OpenAPI Amazon AWS | S3

Amazon Simple Storage Service (Amazon S3) は、業界をリードするスケーラビリティ、データ可用性、セキュリティ、パフォーマンスを提供するオブジェクトストレージサービスです。あらゆる規模および業界のお客様が、データレイク、ウェブサイト、モバイルアプリケーション、バックアップと復元、アーカイブ、企業アプリケーション、IoT デバイス、ビッグデータ分析など、さまざまなユースケースに対して、任意の量のデータを保存および保護するために Amazon S3 を使用できます。Amazon S3 は、特定のビジネス、組織、コンプライアンス要件を満たすようにデータへのアクセスを最適化、整理、設定するための管理機能を提供します。

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 クライアント (TsgcOpenAPI_Google_Client) は、[TsgcOpenAPI_Client](#) から継承した独自の OpenAPI クライアントを持っています。

このコンポーネントには、Google Cloud サーバーへの接続に必要なすべての設定を含む **GoogleOptions** というプロパティがあります。

GoogleOptions

OpenAPI Google クライアントでは以下の方法で認証できます:

1. **OAuth2 Code**: インタラクティブです。つまり、ユーザーの介入が必要です。
2. **JWT (サービスアカウント)**: 非インタラクティブなので、たとえばサービスとして実行できます。

認証は `GoogleOptions.Authentication` プロパティで設定され、以下の値を使用できます。

- **oagaOAuth2**: インタラクティブ。
- **oagaJWT**: 非インタラクティブ。JSON ファイルからメソッド `LoadSettingsFromFile` を使用して設定をインポートできます。このメソッドは以下のプロパティを自動的に入力します:
 - **ClientEmail**
 - **PrivateKeyId**
 - **PrivateKey**
 - **ServiceAccountOptions**: サービスアカウントを使用してクライアントを実行する場合、以下のプロパティが必要です:
 - **TokenURI**: デフォルト値は "https://oauth2.googleapis.com/token" ですが、`LoadSettingsFromFile` メソッドを使用するときに値が更新されます。
 - **Subject**: ドメイン全体の委任を使用する際のメールアカウントです
 - **Scopes**: スコープのリスト。

最も一般的な使用方法

- **設定**
 - [Google Cloud OAuth2](#)
 - [Google Cloud サービスアカウント](#)
- **API**
 - [Google Cloud PubSub](#)
 - [Google Cloud Calendar](#)
 - [Google Cloud Drive](#)

sgcOpenAPI Google Cloud SDK

現在利用可能な API のリストを以下に示します。

- 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 デバイスプロビジョニングパートナー API
- Android Management API
- API ディスカバリーサービス
- 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 管理 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
- クラウド 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
- クラウドプライベートカタログ
- 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
- データパイプライン 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 ライセンスマネージャー API
- エラーレポート API
- Essential Contacts API
- Eventarc API
- ファクトチェックツール API
- Firebase App Check API
- Firebase Cloud Messaging API
- Firebase Cloud Messaging データ 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 カスタムアプリ公開 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 リセラー API
- Google+ API
- Groups Migration API
- グループ設定 API
- HomeGraph API
- IAM Service Account Credentials API
- Idea Hub API
- Identity and Access Management (IAM) API
- Indexing API
- Knowledge Graph 検索 API
- Kubernetes Engine API
- Library Agent API
- ローカルサービス API
- Managed Service for Microsoft Active Directory API
- Manufacturer Center API
- マイビジネスアカウント管理 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
- ネットワーク管理 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 (ベータ)
- Recommender API
- Remote Build Execution API
- レプリカプール
- Resource Settings API
- Retail API
- Safe Browsing API
- SAS Portal API
- SAS Portal API (テスト)
- Search Ads 360 API
- Search Console API
- Secret Manager API
- Security Command Center API
- Security Token Service API
- Semantic Tile API
- Service Broker
- サービスコンシューマー管理 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
- バージョン履歴 API
- VM Migration API
- Web Fonts 開発者 API
- Web Risk API
- Web Security Scanner API
- ワークフロー実行API
- Workflows API
- YouTube Analytics API
- YouTube Data API v3
- YouTube Reporting API

OpenAPI Google Cloud | OAuth2

OpenAPI Google Cloudコンポーネントを使用してOAuth2で認証するには、まずGoogle CloudからOAuth2 Keyを取得する必要があります。

Google OAuth2 キーを取得する手順と、PubSub サンプルアプリケーションでの設定方法を以下に示します。

Google Cloud アカウントに最初にログインし、既存のプロジェクトを使用するか新しいプロジェクトを作成します。

その後、**認証情報**メニューに移動し、**認証情報を作成**ボタンをクリックして、オプション **OAuth クライアント ID** を選択します。

アプリケーションタイプを選択し、説明名を設定します

成功すると、クライアントIDとクライアントシークレットが取得されます。

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

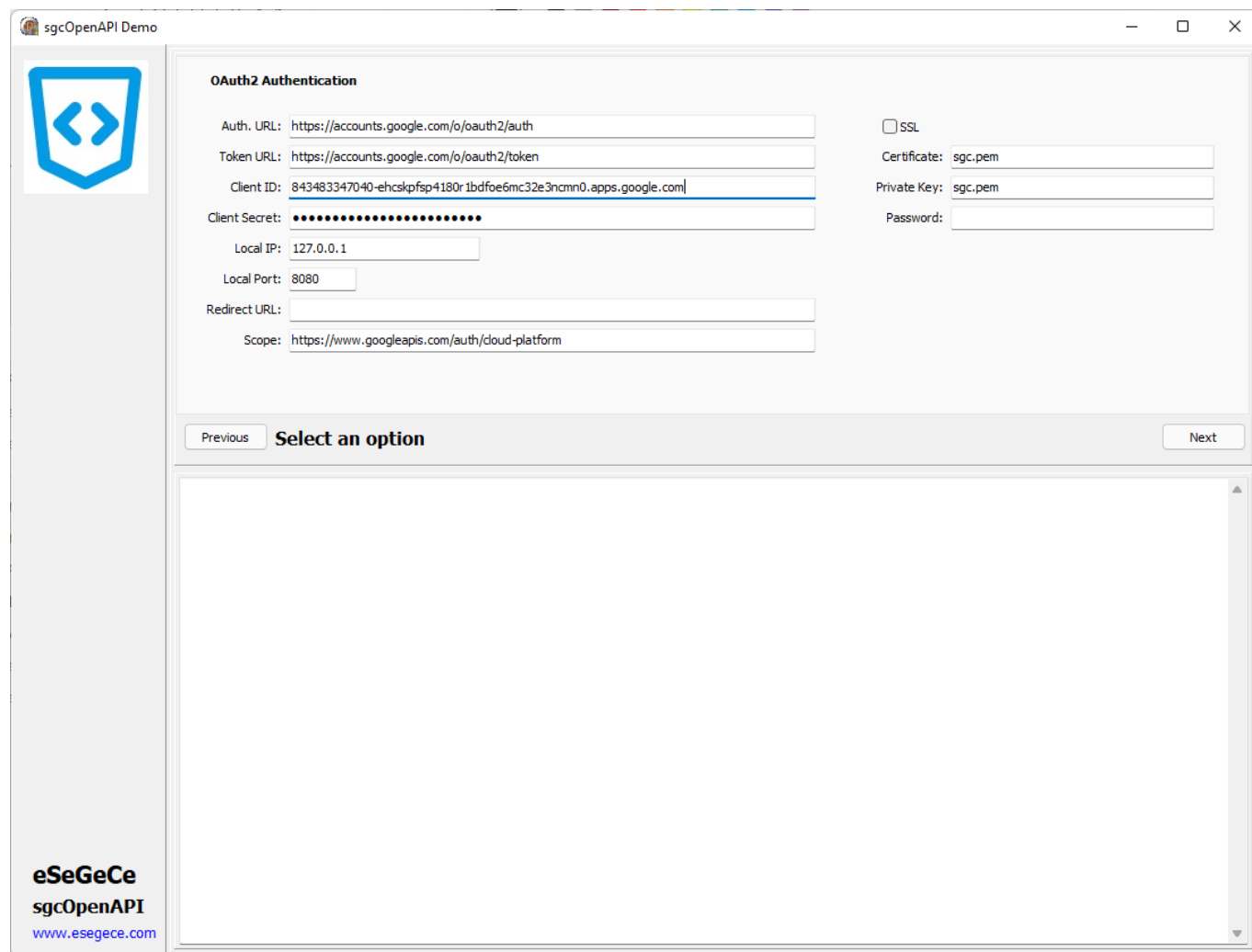
pvogD9reE0t9ilL6eR1jE60Z



OK

OAuth2 データを誰とも共有しないでください！

OpenAPI Google Cloud サンプルにコピーします。



sgcOpenAPI Demo

OAuth2 Authentication

Auth. URL:

Token URL:

Client ID:

Client Secret:

Local IP:

Local Port:

Redirect URL:

Scope:

☐ SSL

Certificate:

Private Key:

Password:

Select an option

eSeGeCe
sgcOpenAPI
www.esegece.com

[OAuth2 設定](#)の詳細をご確認ください。

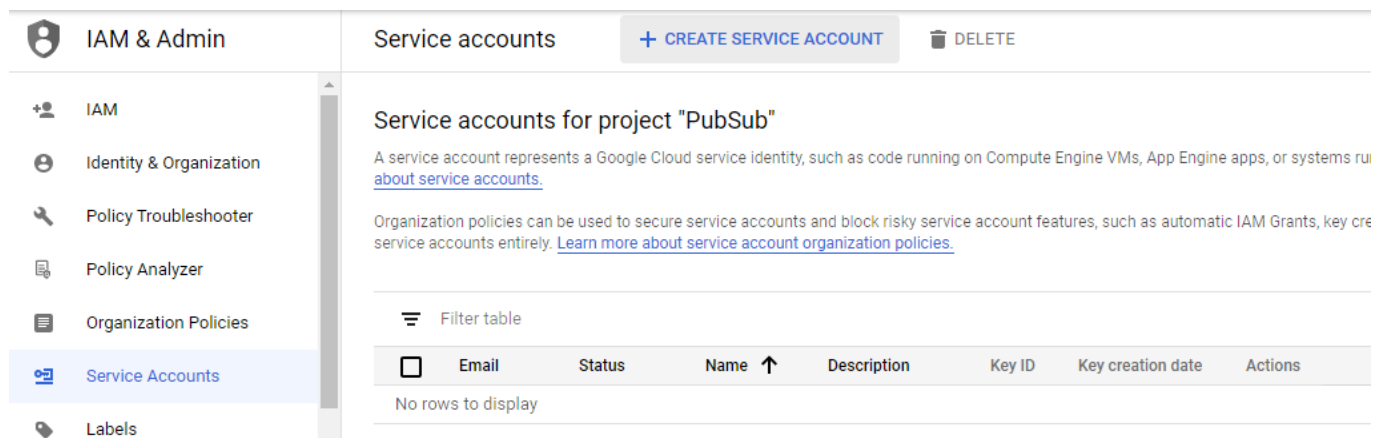
認証が完了したら、まず**ClearOAuth2Token** メソッド（内部のすべてのOAuth2トークンをクリアします）を呼び出してから 任意のOpenAPIリクエストを呼び出すことで、**再認証**できます。googleサーバーに対して再認証するための新しいWebブラウザーが表示されます。

OpenAPI Google Cloud | サービスアカウント

OpenAPI Google Cloud コンポーネントを使用してサービスアカウントで認証するには、まず Google Cloud から秘密鍵証明書を取得する必要があります。

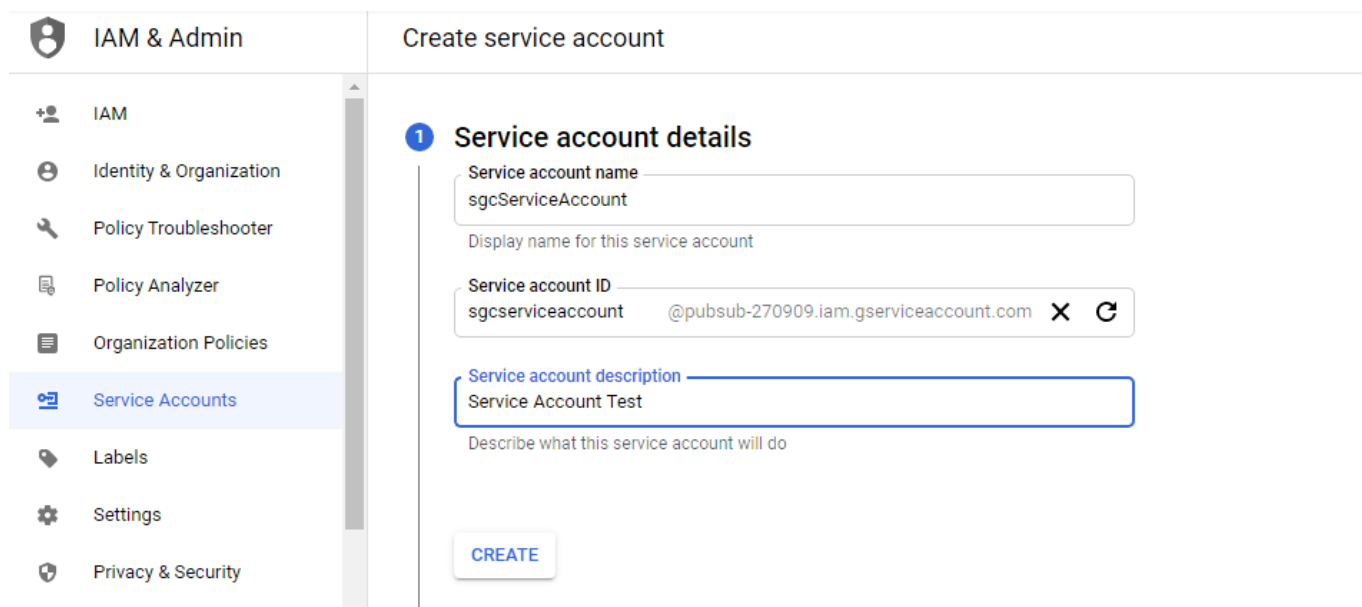
Google プライベートキー証明書の取得方法と、サンプルアプリケーションでの設定方法について以下に示します。

Google Cloud アカウントに最初にログインし、既存のプロジェクトを使用するか新しいプロジェクトを作成します。



The screenshot shows the 'IAM & Admin' section of the Google Cloud console. The left sidebar lists various tools, with 'Service Accounts' highlighted. The main content area is titled 'Service accounts for project "PubSub"' and includes a '+ CREATE SERVICE ACCOUNT' button. Below this, there is explanatory text about service accounts and a table with columns: Email, Status, Name, Description, Key ID, Key creation date, and Actions. The table currently shows 'No rows to display'.

CREATE SERVICE ACCOUNT を選択すると新しいページが表示され、サービスアカウント名と説明を設定する必要があります



The screenshot shows the 'Create service account' page. The left sidebar is the same as the previous image. The main content area is titled 'Create service account' and contains a '1 Service account details' section. This section has three input fields: 'Service account name' (filled with 'sgcServiceAccount'), 'Service account ID' (filled with 'sgcServiceAccount' and a domain '@pubsub-270909.iam.gserviceaccount.com'), and 'Service account description' (filled with 'Service Account Test'). A 'CREATE' button is at the bottom.

次に、少なくとも1つのRoleを選択します。クライアントがトピックをパブリッシュおよびサブスクライブできるようにPubSub Adminを選択しますが、より少ない権限を持つ別のロールを選択することもできます

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](#)

続行を押して、最終的に他のユーザーへのアクセスを許可できます

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](#)

完了したら DONE を押すと、新しいレコードが表示されます。

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

次のステップは新しいKeyを作成することです。actions列でCreate Keyオプションを選択します。JSONを選択してJSON形式の構成をダウンロードすると、新しいKeyが作成されます

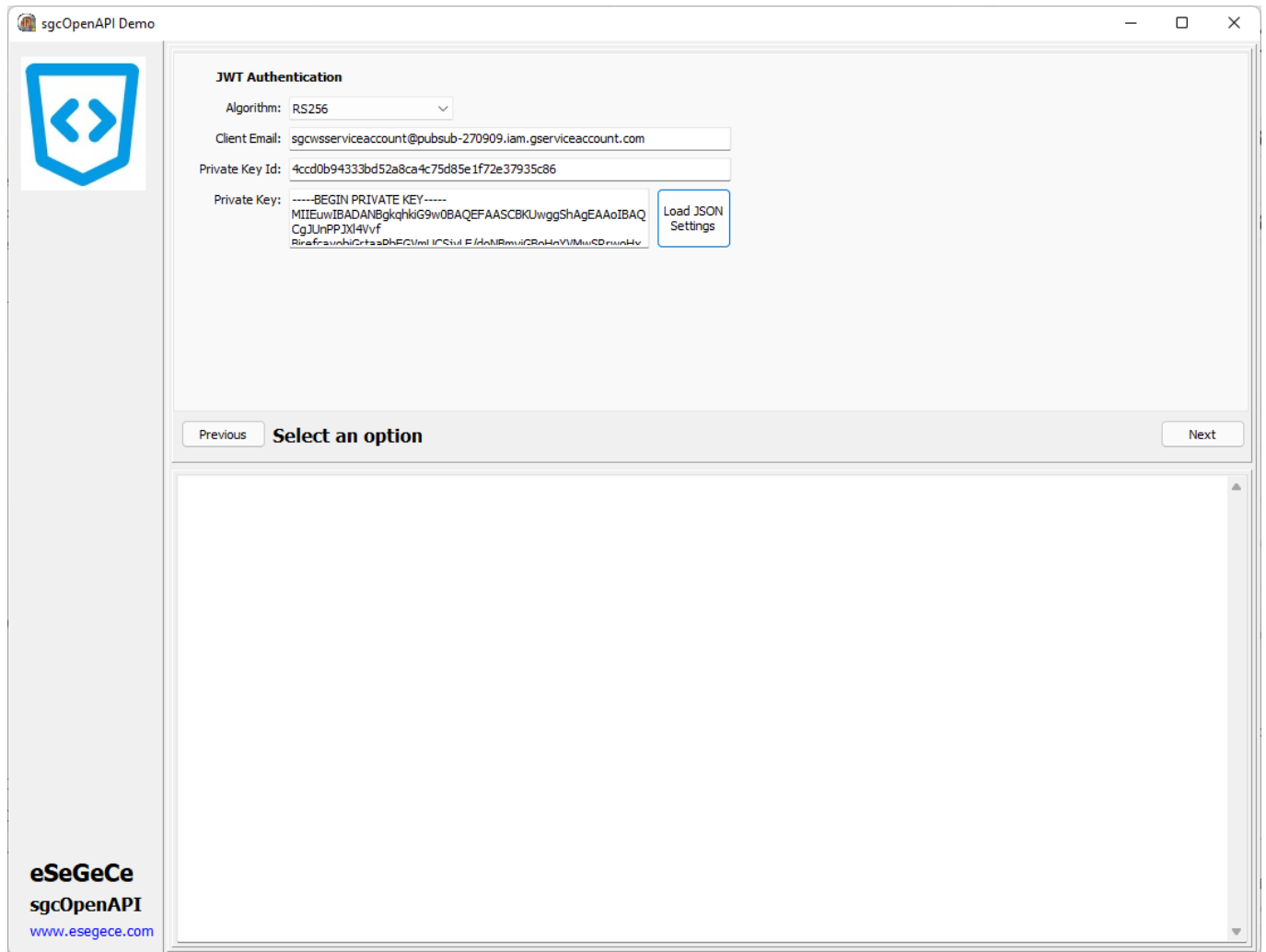
Service accounts for project "PubSub"

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	⋮

最後に、Google が提供するデータを OpenAPI PubSub クライアントに入力するだけです。設定 JSON ファイルを読み込むには **LoadSettingsFromFile** を使用できます。



sgcOpenAPI Demo

JWT Authentication

Algorithm: RS256

Client Email: sgcserviceaccount@pubsub-270909.iam.gserviceaccount.com

Private Key Id: 4ccd0b9433bd52a8ca4c75d85e1f72e37935c86

Private Key: -----BEGIN PRIVATE KEY-----
 MIIEUwIBADANBgkqhkiG9w0BAQEFAASCBCUwggShAgEAAoIBAQCgJUnPPJXh4Vvf
 BirafrunhiCrtasDhECUwIIRStU FIdnIRmuitCnHnYUmuSDrunhV

Load JSON Settings

Previous **Select an option** Next

eSeGeCe
 sgcOpenAPI
www.esegece.com

ドメイン全体の委任

Google Workspaceアカウントをお持ちの場合、組織の管理者はGoogle Workspaceドメインのユーザーに代わってユーザーデータにアクセスするアプリケーションを承認できます。たとえば、**Google Calendar API**を使用してGoogle Workspaceドメインのすべてのユーザーのカレンダーにイベントを追加するアプリケーションは、ユーザーに代わってGoogle Calendar APIにアクセスするためにサービスアカウントを使用します。ドメイン内のユーザーのデータにアクセスするためにサービスアカウントを承認することは、「ドメイン全体の委任」とも呼ばれます。

サービスアカウントにドメイン全体の権限を委任するには、Google Workspace ドメインのスーパー管理者が次の手順を実行する必要があります：

- Google Workspace ドメインの管理コンソールで、**メインメニュー > セキュリティ > アクセスとデータ制御 > API コントロール**に移動します。
- **Domain wide delegation** ペインで **Manage Domain Wide Delegation** を選択します。
- **新規追加** をクリックします。
- **Client ID** フィールドに、サービスアカウントの **Client ID** を入力します。サービスアカウントのクライアント ID は「サービスアカウント」ページで確認できます。
- **OAuth スコープ (カンマ区切り)** フィールドに、アプリケーションがアクセスを許可されるスコープのリストを入力します。例えば、アプリケーションが Google Drive API と Google Calendar API にドメイン全体でのフルアクセスを必要とする場合は、<https://www.googleapis.com/auth/drive>, <https://www.googleapis.com/auth/calendar> と入力します。
- **Authorize** をクリックします。

ワークスペースアカウントをリンクして認証したら、OpenAPI クライアントのプロパティ `GoogleOptions.ServiceAccountOptions` を設定してください。

- **Subject:** サービスアカウントにリンクされたワークスペースのメールアカウントです。例：
youremail@domain.com
- **Scopes:** スコープのリスト。例: <https://www.googleapis.com/auth/calendar>
- **TokenURI:** デフォルトは <https://oauth2.googleapis.com/token> です。

OpenAPI Google Cloud | PubSub

Pub/Subはサービスが100ミリ秒オーダーのレイテンシで非同期に通信できるようにします。

Pub/Sub はデータの取り込みと配信のためのストリーミング分析とデータ統合パイプラインに使用されます。サービス統合のためのメッセージ指向ミドルウェアとして、またはタスクを並列化するためのキューとしても同様に効果的です。

List Projects by Topic (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");
```

トピックによるプロジェクトの一覧取得（サービスアカウント）

```
GetOpenAPIClient->GoogleOptions->Authentication = oagaJWT;  
GetOpenAPIClient->LoadSettingsFromFile("google.json");  
GetOpenAPIClient->ListV1TopicsByProject("projects/pubsub-270909");
```


OpenAPI Google Cloud Calendar

Calendar API を使用することで、アプリを Google Calendar と統合し、ユーザーとの新しいエンゲージメント方法を作成できます。

カレンダーごとのイベント一覧 (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, "");
```

カレンダーごとのイベント一覧 (サービスアカウント)

```
GetOpenAPIClient->GoogleOptions->Authentication = oagaJWT;
GetOpenAPIClient->LoadSettingsFromFile("google.json");
GetOpenAPIClient->Authentication->ServiceAccountOptions->Subject = "email@mydomain.com";
GetOpenAPIClient->Authentication->ServiceAccountOptions->Scopes->Text := "https://www.googleapis.com/auth/calendar";
GetOpenAPIClient->ListCalendarsEventsByCalendarId("email@mydomain.com", true, "json", 0, "");
```

カレンダーイベントの挿入

```
GetOpenAPIClient->InsertCalendarsEventsByCalendarId("email@mydomain.com", "{ \"summary\": \"Test Event\", \"description\": \"\" }", true, "json", 0, "");
```

OpenAPI Google Cloud | Drive

Google Drive アカウトにファイルをアップロード/ダウンロードする場合は、sgcOpenAPI_googleapis.com_drive_helpers ユニットを使用してください。

ファイルをアップロードする

Google ドライブアカウントに新しいファイルをアップロードするには UploadFile メソッドを使用してください。**OnUpload** イベントでアップロードの進捗を確認できます。

```
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();
}
```

ファイルのダウンロード

DownloadFileメソッドを使用して、Google Driveアカウントからファイルをダウンロードします。**OnDownload** イベントを使用すると、ダウンロードの進行状況を確認できます。

```
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();
}

```

フォルダーの一覧表示

ドライブアカウント内のすべてのフォルダを一覧表示します。**ListFiles** メソッドを使用し、パラメーター q として「application/vnd.google-apps.folder」を渡します。

```

GetOpenAPIClient().ListFiles("", true, "", false, false, 100, "", false, false, "", "", "", "", "", "", "", "", "

```

OpenAPI | Microsoft

sgcOpenAPI Microsoft クライアント (TsgcOpenAPI_Microsoft_Client) には、[TsgcOpenAPI_Client](#) から継承する専用の OpenAPI クライアントがあります。

このコンポーネントには Microsoft サーバーへの接続に必要なすべての設定を含む **MicrosoftOptions** プロパティがあります。

MicrosoftOptions

OpenAPI Microsoft クライアントでは以下の方法で認証できます:

1. **OAuth2 Code**: インタラクティブです。つまり、ユーザーの介入が必要です。
2. **OAuth2 Credentials**: 非インタラクティブです。例えばサービスとして実行できます。

認証は `MicrosoftOptions.Authentication` プロパティで設定され、次の値を使用できます:

- **oamaOAuth2Code**: インタラクティブ。
- **oamaOAuth2Credentials**: 非インタラクティブ。

Microsoft が必要とするその他のプロパティは以下のとおりです。

- **TenantId**: アカウントを識別する値です。Microsoft/Azure アカウントで確認できます。

最も一般的な使用方法

- **設定**
 - [Microsoft テナントの取得](#)
 - [Microsoft アプリケーションの登録](#)
 - [Microsoft OAuth2 コード](#)
 - [Microsoft OAuth2 認証情報](#)
- **APIs**
 - [Microsoft Graph](#)

sgcOpenAPI Microsoft APIs

現在利用可能な API のリストを以下に示します。

- 自動提案クライアント
- Computer Vision クライアント
- カスタム画像検索クライアント
- Custom Search クライアント
- Custom Vision Prediction クライアント
- Custom Vision Training Client
- Entity Search Client
- 画像検索クライアント
- ローカル検索クライアント
- News Search クライアント
- Partial Graph API
- スペルチェッククライアント
- Video Search クライアント
- Visual Search Client
- Web Search Client

sgcOpenAPI Azure APIs

現在利用可能な API のリストを以下に示します。

- API Client
- ACE Provisioning ManagementPartner
- ADHybridHealthService
- AdvisorManagementClient
- Anomaly Detector クライアント
- 異常検出クライアント
- ApiManagementClient
- AppConfigurationManagementClient
- Application Insights Data Plane
- ApplicationClient
- ApplicationInsightsManagementClient
- AppPlatformManagementClient
- AppServiceCertificateOrders API クライアント
- AppServiceEnvironments API クライアント
- AppServicePlans API クライアント
- アーティファクト
- AttestationClient
- AuthorizationManagementClient
- AutomationManagement
- AutomationManagementClient
- Azure Action Groups
- Azure Activity Log アラート
- Azure Addons Resource Provider
- Azure Alerts Management Service リソースプロバイダー
- Azure Bot Service
- Azure CDN WebApplicationFirewallManagement
- Azure Data Catalog Resource Provider
- Azure Data Lake Storage
- Azure Data Migration Service リソースプロバイダー
- 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 クエリパック
- Azure Machine Learning Datastore Management Client
- Azure Machine Learning モデル管理サービス
- Azure Machine Learning ワークスペース
- Azure Maps リソースプロバイダー
- Azure Media Services
- Azure Metrics
- Azure Migrate Hub
- Azure Migrate V2
- Azure ML Commitment Plans Management クライアント
- Azure ML Web サービス管理クライアント
- Azure Monitor プライベートリンクスコープ
- Azure Reservation
- Azure Resource Graph
- Azure Resource Graph クエリ
- Azure SQL Database
- Azure SQL Database API 仕様
- Azure SQL データベースのバックアップ
- Azure SQL Database Backup Long Term Retention Policy
- Azure SQL Database データマスキングポリシーとルール
- Azure SQL Database 障害復旧構成
- Azure SQL データベースのインポート/エクスポート仕様
- Azure SQL Database レプリケーションリンク
- Azure SQL Server API 仕様
- Azure SQL Server バックアップ長期保持ボールド

- Azure Stack Azure Bridge クライアント
- azureactivedirectory
- AzureAnalysisServices
- AzureBridgeAdminClient
- AzureDataManagementClient
- AzureDeploymentManager
- AzureDigitalTwinsManagementClient
- AzureStack Azure Bridge クライアント
- BackupManagementClient
- BatchAI
- BatchManagement
- BatchService
- BillingManagementClient
- BlockchainManagementClient
- BlueprintClient
- CdnManagementClient
- CertificateRegistrationProvider API クライアント
- 証明書 API クライアント
- CognitiveServicesManagementClient
- CommerceManagementClient
- Compute Admin クライアント
- ComputeDiskAdminManagementClient
- ComputeManagementClient
- ComputeManagementConvenienceClient
- コンピュータービジョン
- 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 クライアント
- DeploymentAdminClient
- DeploymentScriptsClient
- DeviceServices
- DevSpacesManagement
- DevTestLabsClient
- 診断 API クライアント
- DiskResourceProviderClient
- DnsManagementClient
- ドメインサービスリソースプロバイダー
- DomainRegistrationProvider API クライアント
- Domains API クライアント
- Dynamics テレメトリ
- Engagement.ManagementClient
- EngagementFabric
- EventGridManagementClient
- EventHub2018PreviewManagementClient
- EventHubManagementClient
- Execution Service

- ExpressRouteCrossConnection REST APIs
- FabricAdminClient
- Face Client
- FeatureClient
- Form Recognizer Client
- FrontDoorManagementClient
- GalleryManagementClient
- ゲスト診断設定
- ゲスト診断設定の関連付け
- GuestConfiguration
- HanaManagementClient
- HDInsightJobManagementClient
- HDInsightManagementClient
- HealthcareApisClient
- HybridComputeManagementClient
- HybridDataManagementClient
- HyperDrive
- InfrastructureInsightsManagementClient
- Ink Recognizer クライアント
- InstanceMetadataClient
- IntuneResourceManagementClient
- iotDpsClient
- iotHubClient
- IoTSpacesClient
- KeyVaultClient
- KeyVaultManagementClient
- KustoManagementClient
- LogicAppsManagementClient
- LogicManagementClient
- LUIS Authoring Client
- LUIS Programmatic
- Machine Learning Compute Management Client
- Machine Learning ワークスペース管理クライアント
- MaintenanceManagementClient
- ManagedLabsClient
- ManagedNetworkManagementClient
- ManagedServiceIdentityClient
- ManagedServicesClient
- 管理グループ
- ManagementLinkClient
- ManagementLockClient
- MariaDBManagementClient
- Marketplace RP サービス
- MarketplaceOrdering.Agreements
- MediaServicesManagementClient
- Microsoft Insights
- Microsoft NetApp
- Microsoft Storage Sync
- Microsoft.ResourceHealth
- Microsoft.Support
- MicrosoftSerialConsoleClient
- Mixed Reality
- ML Team アカウント管理クライアント
- MonitorManagementClient
- MySQLManagementClient
- NetworkAdminManagementClient
- NetworkExperiments
- NetworkManagementClient
- NotificationHubsManagementClient
- PeeringManagementClient
- Personalizer クライアント
- PolicyClient
- PolicyEventsClient
- PolicyMetadataClient
- PolicyStatesClient

- PolicyTrackedResourcesClient
- portal
- PostgreSQLManagementClient
- Power BI Embedded Management Client
- PowerBIDedicated
- PrivateDnsManagementClient
- プロバイダー API クライアント
- QnAMaker Client
- QnAMaker Runtime Client
- Recommendations API クライアント
- RecoveryServicesBackupClient
- RecoveryServicesClient
- RedisManagementClient
- リレー
- RemediationsClient
- ResourceHealthMetadata API クライアント
- ResourceManagementClient
- Run History APIs
- RunCommandsClient
- SchedulerManagementClient
- SeaBreezeManagementClient
- SearchIndexClient
- SearchManagementClient
- SearchServiceClient
- Security Center
- セキュリティインサイト
- ServerManagement
- Service Fabric クライアント API
- サービスマップ
- ServiceBusManagementClient
- ServiceFabricManagementClient
- SharedImageGalleryServiceClient
- SignalRManagementClient
- SiteRecoveryManagementClient
- Software Plan RP
- SqlManagementClient
- SqlVirtualMachineManagementClient
- Storage Cache Mgmt Client
- StorageImportExport
- StorageManagementClient
- StorSimple8000SeriesManagementClient
- StorSimpleManagementClient
- StreamAnalyticsManagementClient
- SubscriptionClient
- SubscriptionDefinitionsClient
- SubscriptionsManagementClient
- Text Analytics クライアント
- TimeSeriesInsightsClient
- TopLevelDomains API クライアント
- TrafficManagerManagementClient
- 更新管理
- UpdateAdminClient
- UsageManagementClient
- VirtualMachineImageTemplate
- VirtualWANAsAServiceManagementClient
- Visual Studio Projects Resource Provider Client
- Visual Studio Resource Provider クライアント
- VM Insights オンボーディング
- VMwareCloudSimple
- WebApplicationFirewallManagement
- WebApps APIクライアント
- WebSite Management クライアント
- windowsesu
- WorkbookClient
- ワークロードモニター

OpenAPI Microsoft | テナント

ID管理とアクセス管理にMicrosoft identity platformを使用するアプリを構築するには、Azure Active Directory (Azure AD) テナントへのアクセスが必要です。アプリの登録と管理、Microsoft 365およびその他のWeb APIのデータへのアクセス設定、条件付きアクセスなどの機能の有効化を行うのは、このAzure ADテナント内です。

テナントは組織を表します。Azure AD の専用インスタンスで、組織またはアプリ開発者が Microsoft との関係を開始したときに受け取ります。その関係は Azure、Microsoft Intune、または Microsoft 365 へのサインアップから始まる場合があります。

各 Azure AD テナントは他の Azure AD テナントとは独立して分離されています。テナントには独自の職場および学校 ID、消費者 ID (Azure AD B2C テナントの場合)、およびアプリ登録があります。テナント内のアプリ登録は、テナント内のアカウントまたはすべてのテナントからの認証のみを許可できます。

既存の Azure AD テナントを使用する

テナントを確認するには:

1. Azure Portalにサインインします。アプリケーションの管理に使用するアカウントを使用してください。
2. 右上隅を確認します。テナントがある場合は、自動的にサインインされます。アカウント名の直下にテナント名が表示されます。

アカウントにテナントが関連付けられていない場合、アカウント名の下に GUID が表示されます。Azure AD テナントを作成するまで、アプリの登録などのアクションを実行することはできません。

新しい Azure AD テナントを作成する

新しいテナントを作成するために次の情報を提供してください:

- **組織名**
- **初期ドメイン** - 初期ドメイン <domainname>.onmicrosoft.com は編集または削除できません。後でカスタマイズされたドメイン名を追加することができます。
- **国または地域**

OpenAPI Microsoft | アプリケーションの登録

アプリケーションを登録することで、アプリと Microsoft ID プラットフォームの間に信頼関係が確立されます。この信頼関係は一方です。アプリは Microsoft ID プラットフォームを信頼しますが、その逆はありません。

アプリ登録を作成するには、次の手順に従ってください。

1. Azure ポータルにサインインします。
2. 複数のテナントにアクセスできる場合は、上部メニューの **ディレクトリ + サブスクリプション** フィルターを使用して、アプリケーションを登録したいテナントに切り替えます。
3. **Azure Active Directory**を検索して選択します。
4. **管理**の下で、**アプリの登録** > **新規登録**を選択します。
5. アプリケーションの表示 **名前**を入力してください。アプリケーションのユーザーは、サインイン時などにこの表示名を見る可能性があります。表示名はいつでも変更でき、複数のアプリ登録が同じ名前を共有することもできます。アプリ登録の自動生成されたアプリケーション（クライアント）ID は、表示名ではなく、ID プラットフォーム内でアプリを一意に識別します。
6. このアプリケーションを誰が使用できるかを指定します。これは **サインインの対象**と呼ばれることもあります。
7. 最初のアプリ登録を完了するには **Register** を選択します

Register an application - Microsoft

https://portal.azure.com

Microsoft Azure Search resources, services, and docs (G+/)

meganb@contoso.com CONTOSO AD (DEV)

Home > Contoso AD (dev) >

Register an application

* Name

The user-facing display name for this application (this can be changed later).

Supported account types

Who can use this application or access this API?

☒ Accounts in this organizational directory only (Contoso AD (dev) only - Single tenant)

☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant)

☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)

☐ Personal Microsoft accounts only

[Help me choose...](#)

Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web e.g. https://myapp.com/auth

By proceeding, you agree to the [Microsoft Platform Policies](#)

Register

リダイレクト URI を追加します

A リダイレクト URI は、Microsoft ID プラットフォームがユーザーのクライアントをリダイレクトし、認証後にセキュリティトークンを送信する場所です。

たとえば、本番環境のウェブアプリケーションでは、リダイレクト URI は通常 `https://contoso.com/auth-response` のようなアプリが実行されているパブリックエンドポイントです。開発中は `https://127.0.0.1/auth-response` や `http://localhost/auth-response` のようなローカルで実行されているエンドポイントも追加するのが一般的です。

この **RedirectURI** は後で **sgcOpenAPI Microsoft クライアント** を設定するために使用されます。

クレデンシャルを追加する

資格情報は、Web APIにアクセスする機密クライアントアプリケーションで使用されます。機密クライアントの例としては、Webアプリ、他のWeb API、またはサービス型・デーモン型アプリケーションがあります。資格情報により、アプリケーションは実行時にユーザーの操作を必要とせず、自身として認証できます。

証明書とクライアントシークレット (文字列) の両方を、機密クライアントアプリ登録の資格情報として追加できます。

The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with 'Microsoft Azure' and a search bar. Below it, the breadcrumb trail is 'Home > Fourth Coffee > Contoso App 1'. The main heading is 'Contoso App 1 | Certificates & secrets'. On the left, a sidebar lists various management options: Overview, Quickstart, Integration assistant, Manage (with sub-items: Branding, Authentication, Certificates & secrets, Token configuration, API permissions, Expose an API, App roles, Owners, Roles and administrators | Preview, Manifest), and Support + Troubleshooting (with sub-items: Troubleshooting, New support request). The 'Certificates & secrets' option is highlighted with a red box. 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. A description states: 'A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.' There is a '+ New client secret' button. Below this is a table with headers: 'Description', 'Expires', 'Value', and 'Secret ID'. The table is currently empty, with a message at the bottom: 'No client secrets have been created for this application.'

クライアントシークレットを追加する

アプリケーションパスワードと呼ばれることもあるクライアントシークレットは、証明書の代わりにアプリが自身を識別するために使用できる文字列値です。

クライアントシークレットは証明書資格情報よりも安全性が低いと考えられています。アプリケーション開発者は使いやすさのためにローカルアプリ開発中にクライアントシークレットを使用することがありますが、本番環境で実行されるアプリケーションには証明書資格情報を使用すべきです。

1. Azure ポータルの **アプリの登録**で、アプリケーションを選択します。
2. **Certificates & secrets > Client secrets > New client secret**を選択します。
3. クライアントシークレットの説明を追加します。
4. シークレットの有効期限を選択するか、カスタムの有効期間を指定してください。
 - クライアントシークレットの有効期間は最大 2 年（24 ヶ月）以内に制限されています。24 ヶ月を超えるカスタム有効期間は指定できません。
 - Microsoft は 12 ヶ月未満の有効期限値を設定することを推奨しています。
5. **追加**を選択します。
6. シークレットの値を記録します。クライアントアプリケーションコードで使用するためです。このシークレット値は、このページを離れると二度と表示されません。

OpenAPI Microsoft | OAuth2 コード

OAuth2 コードグラントフローを使用するには、ログインして必要な権限を取得するためにユーザーとのやり取りが必要です。

Tenant Idと資格情報を入手したら、Code Grant用のOAuth2プロパティを構成できます。

OAuth2 Code Grant用にOpenAPI Clientを構成するには、**MicrosoftOptions.Authentication**プロパティを次の値で構成します:

```
GetOpenAPIClient->MicrosoftOptions->Authentication = oamaOAuth2Code;
```

次に OAuth2 パラメーターを設定できます

```
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  
";
```

テナント ID は認証とトークンの URL に設定する必要があります。URL の正しいテナント ID を置き換えてください。

Microsoftはローカルエイリアスでリッスンしている場合のみURLとして**localhost**を許可しているため、IPアドレスで設定する代わりにDNS名としてlocalhostを使用して**リダイレクトURL**を設定してください。

scope の値はAPIによって異なります。各APIについてはMicrosoft / Azureのドキュメントを確認してください。

初回リクエスト時は、ユーザーが Microsoft アカウントにログインするよう求めるウェブブラウザが表示されます。ユーザーが以前にログインしている場合は、HTTP リクエストが直接実行されます。

OpenAPI Microsoft | OAuth2 クレデンシャル

OAuth2 コードグラントフローの使用はユーザーとのインタラクションを必要としないため、サービス、デーモン、またはユーザーとのインタラクションなしに実行する必要があるアプリケーションに適しています。

Tenant Idと資格情報を入手したら、Code Grant用のOAuth2プロパティを構成できます。

OAuth2 Code Grant用にOpenAPI Clientを構成するには、**MicrosoftOptions.Authentication**プロパティを次の値で構成します:

```
GetOpenAPIClient->MicrosoftOptions->Authentication = oamaOAuth2Credentials;
```

OAuth2 Authentication

Auth. URL:

Token URL:

Client ID:

Client Secret:

Local IP:

Local Port:

Redirect URL:

Scope:

☒ SSL

Certificate:

Private Key:

Password:

Select an option

eSeGeCe
sgcOpenAPI
www.esegece.com

```
GetOpenAPIClient->MicrosoftOptions->Authentication = oamaOAuth2Credentials;
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->AuthURL = "https://login.microsoftonline.com/oauth2/v2.0/authorize";
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->TokenURL = "https://login.microsoftonline.com/oauth2/v2.0/token";
GetOpenAPIClient->Authentication->OAuth2->OAuth2Options->ClientId = "90945b8d-f6b7-4b97-b2bd-21c3c90b5f3b";
GetOpenAPIClient->Authentication->OAuth2->OAuth2Options->ClientSecret = "client_secret";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->IP = "127.0.0.1";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->Port = 8080;
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSL = True;
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSLOptions->Port = 8080;
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSLOptions->CertFile = "sgc.pem";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSLOptions->KeyFile = "sgc.pem";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->SSLOptions->Password = "";
GetOpenAPIClient->Authentication->OAuth2->LocalServerOptions->RedirectURL = "https://localhost:8080";
GetOpenAPIClient->Authentication->OAuth2->AuthorizationServerOptions->Scope->Text = "openid";
```

```
https://graph.microsoft.com/.default
```

```
";
```

テナント ID は認証とトークンの URL に設定する必要があります。URL の正しいテナント ID を置き換えてください。

Microsoftはローカルエイリアスでリッスンしている場合のみURLとして**localhost**を許可しているため、IPアドレスで設定する代わりにDNS名としてlocalhostを使用して**リダイレクトURL**を設定してください。

scope の値はAPIによって異なります。各APIについてはMicrosoft / Azureのドキュメントを確認してください。

OAuth2 認証情報はユーザーインタラクションを必要としないため、最初の HTTP リクエスト呼び出しでブラウザが開かれることはありません。

OpenAPI Microsoft | Graph

Microsoft Graph は、次の Microsoft クラウドサービスのデータにアクセスするための REST API とクライアントライブラリを公開しています:

- Microsoft 365 コアサービス: Bookings、Calendar、Delve、Excel、Microsoft 365 コンプライアンス eDiscovery、Microsoft Search、OneDrive、OneNote、Outlook/Exchange、People (Outlook 連絡先)、Planner、SharePoint、Teams、To Do、Workplace Analytics
- Enterprise Mobility + Securityサービス: Advanced Threat Analytics、Advanced Threat Protection、Azure Active Directory、Identity Manager、Intune
- Windows サービス: アクティビティ、デバイス、通知、ユニバーサル印刷
- Dynamics 365 Business Central

現在のユーザーを取得する

```
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();
```

API

以下のAPIはeSeGeCe sgcOpenAPIジェネレーターを使用して生成されており、すべてのユーザーに無料で提供されています。インターフェースのソースコードはトライアルに含まれており、Google、Amazon、Microsoftなどのプライベートライセンスを購入した場合に何を期待できるかを確認できます。

API	説明
Geolocation	Abstract IP ジオロケーション API は IP アドレスを受け取り、場所や住所、タイムします。

OpenAPI | AbstractApi ジオロケーション

IP Geolocation API とは何か？

Abstract IP Geolocation APIはIPアドレスを受け取り、場所に変換するとともに、住所、タイムゾーンなど多くの詳細情報を提供します。

IP ジオロケーション API のユースケースにはどのようなものがありますか？

IPジオロケーションAPIとデータには、多くの強力なユースケースがあります。これらには次のものが含まれますが、これらに限定されません:

- ユーザーの位置情報に基づいて関連するサイトまたはサブサイトに自動的にリダイレクトします
- ユーザーに明示的なカスタマイズを要求せずに、ユーザーの位置情報、国、またはタイムゾーンを自動検出して表示します
- ユーザーの位置情報に基づいてウェブサイトやアプリのコンテンツや体験をカスタマイズします。例: ユーザーの地元の天気、税・VAT 率、通貨、ニュース、祝日などを表示します。
- 例えば、特定の国でサービスを提供できない場合など、ユーザーの場所に基づいてフィルタリングします。
- EU市民向けのGDPRのCookieバナーなど、地域の規制で要求される特定の条件をユーザーが受け入れることを要求します

API キーの取得先はどこですか？

abstractapi.comに登録するだけで、無料のAPIキーが取得できます

<https://www.abstractapi.com/api/ip-geolocation-api>

サンプルコード

以下のコードは、提供されたIPアドレスに関する情報を返します。

```
ShowMessage(GetOpenAPIClient->Retrieve_the_location_of_an_IP_address("asdfkjlkj32i3j2liwj3es", "88.5.12.4"));
```

デモ | サーバーチャット

このデモは、[TsgcWebSocketHTTPServer](#) と WebSocket を通信プロトコルとして使用したサーバーチャットの構築方法を示します。

新しいピアがメッセージを送信するたびに、サーバーはそのメッセージを読み込み、接続されているすべてのクライアントにブロードキャストします。

サーバーの起動

サーバーを起動する前に、リスニングポートの設定やセキュア接続の使用有無などを設定する必要があります。

- まず、[TsgcWebSocketHTTPServer](#)の新しいインスタンスを作成します。
- サーバーがセキュア接続を使用する場合、PEM 証明書が必要です。この証明書の場所と SSL のリスニングポートを設定してください。TLS バージョンと OpenSSL API（必要な場合）を設定できます。

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

- デフォルトでは、サーバーを起動するとリスンポートのすべての IP でリスンします。特定の 1 つの IP のみでリスンするには、バインディングプロパティを使用することをお勧めします。

```
With WSServer->Bindings->Add do
{
    Port = StrToInt(txtDefaultPort->Text);
    IP = txtHost->Text;
}
```

- すべてのオプションを設定したら、`Server.Active = true` を呼び出してサーバーを起動します。

イベント設定

- OnConnect** と **OnDisconnect** イベントを使用して、クライアントがサーバーに接続または切断するタイミングを把握します。

- ・ クライアントからサーバーへ 送信されたメッセージは**OnMessage** イベントで受信されます。このイベントハンドラーを使用して、受信したメッセージを すべてのクライアントにブロードキャストしてください

```
private void OnMessageEvent(TsgcWSConnection *Connection, string Text)
{
    server->Broadcast(Text);
}
```

HTTP リクエストのディスパッチ

WebSocket HTTP Serverを使用すると、同じリッスンポートでWebSocketプロトコルとHTTPプロトコルの両方を処理できるため、WebブラウザがサーバーにアクセスするためにWebページを リクエストできます。**OnCommandGet**は、 HTTPリクエストを読み取り、HTTPレスポンスを送信するために使用されるイベントです。

ARequestInfo パラメーターを使用して HTTP リクエストを読み取り、AResponseInfo を使用して HTTP レスポンスを書き込みます。

基本的に、ARequestInfo.Document を使用してクライアントがリクエストしているドキュメントを読み取り、ResponseNo、ContentType、ContentText プロパティを使用してレスポンスを送信します。

使用例： クライアントがドキュメント 'jquery.js' をリクエストします。

```
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;
    }
}
```

クライアントチャット

このデモでは [TsgcWebSocketClient](#) を使用してクライアントチャットを構築する方法を示します。これは WebSocket サーバーに接続し、メッセージを送信し、このメッセージがすべての接続済みクライアントによって受信されます。

サーバーへ接続

- まず [TsgcWebSocketClient](#) の新しいインスタンスを作成します。
- 次にサーバーの **Host** と **Port** を設定します。
- クライアントがセキュアな接続を使用する場合は、コンポーネントの **TLSOptions** プロパティを設定してください。

```
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;
```

- すべてのオプションを設定したら、`Client.Active = true` を設定してサーバーに接続します。

サーバーへのメッセージ送信

- サーバーにメッセージを送信するには、**WriteData** メソッドを使用し、任意のテキストメッセージを送信すると、サーバーは同じメッセージをレスポンスとして返します。

```
WSClient->WriteData(txtName->Text + ": " + txtMessage->Text);
```

サーバーからのメッセージ受信

- クライアントが新しいテキストメッセージを受信するたびに、**OnMessage** イベントが発生します。

```
private void OnMessageEvent(TsgcWSConnection *Connection, string Text)
{
    DoLog(Text);
}
```

デモ | クライアント

このデモでは、[TsgcWebSocketClient](#) を使用して WebSocket クライアントを構築する方法を示します。

サーバーへ接続

- まず [TsgcWebSocketClient](#) の新しいインスタンスを作成します。
- 次にサーバーの **Host** と **Port** を設定します。
- デフォルトでは、クライアントは **WebSocketプロトコル** を使用して接続します。ただし、**プレーンなTCPプロトコル** を使用して接続するようにクライアントを設定することもできます。
Specifications.RFC6455 = false を設定するだけで、クライアントはWebSocketプロトコルの代わりにプレーンなTCPプロトコルを使用します。詳しくは[TCP Connections](#)をご覧ください。

```
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;
```

クライアントイベント

クライアントのフローを制御するために以下のイベントを使用します：接続時、切断時、メッセージ受信時、エラー検出時など。

```
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);
}
```

デモ | クライアント MQTT

このデモでは MQTT ブローカーサーバーへの接続方法を示します。WebSocket/TCP プロトコルを処理するには [TsgcWebSocketClient](#) が必要です。

設定

- まず新しい [TsgcWebSocketClient](#) インスタンスを作成し、[クライアント デモ](#)を確認してください。
- 次に、[TsgcWSPClient_MQTT](#) の新しいインスタンスを作成します。
- その後、MQTT プロトコルを WebSocket クライアントに割り当て、WebSocket クライアントで接続オプションを設定する必要があります。

```
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

接続フローは MQTT クライアントコンポーネントによって制御されるため、ブローカーへの接続時、新しいメッセージのパブリッシュ時、切断時などを知るために MQTT イベントを処理する必要があります。

```
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;
}
```

デモ | クライアント SocketIO

このデモは Socket.IO サーバーへの接続方法を示しています。WebSocket / TCP プロトコルを処理するために [TsgcWebSocketClient](#) が必要です。

設定

- まず新しい [TsgcWebSocketClient](#) インスタンスを作成し、[クライアント デモ](#)を確認してください。
- 次に、[TsgcWSAPI_SocketIO](#)の新しいインスタンスを作成します。
- その後、Socket.IO API を WebSocket クライアントに割り当て、WebSocket クライアントで接続オプションを設定する必要があります。

```
if (socketio == null)
{
    socketio = new TsgcWSAPI_SocketIO();
    socketio->Client = client;
}
socketio->Client = client;
txtParameters->Text = "/";
chkTLS->Checked = true;
chkOverWebSocket->Checked = true;
```

Send Messages

Socket.IO は [TsgcWebSocketClient](#) を使用してサーバーにメッセージを送信します。**WriteData** を呼び出し、JSON メッセージを socket.io サーバーへのパラメータとして渡します

```
client->WriteData("42[\"new message\", \"\" + txtSocketIOMessage.Text + "\"]");
```

メッセージの受信

接続フローとして受信するメッセージは [TsgcWebSocketClient](#) によって処理されます。このコンポーネントを使用して、サーバーから送信されたメッセージを読み取り、接続がアクティブかどうかを確認してください。

```
private void OnMessageEvent(TsgcWSConnection *Connection, string Text)
{
    DoLog(Text);
}
```

デモ | サーバー モニター

このデモでは、WebSocketをプロトコルとして使用して3つのHTMLモニターを更新する方法を示します。サーバーには内部タイマーがあり、ゲージの値をランダムに更新し、WebSocketメッセージを使用して値を更新します。このメッセージはJavaScriptクライアントによって読み取られ、ゲージの値を更新します。

設定

- まず新しい [TsgcWebSocketServer](#) インスタンスを作成し、[サーバー チャットデモ](#)を確認してください。
- 次に新しいタイマーを作成し、500 ミリ秒ごとにメモリ、ネットワーク、または CPU の値を更新してください。接続されているすべてのクライアントに更新を送信します。
- JavaScript クライアントで、サーバーが送信したメッセージを読み取り、ゲージの値を更新します。

```
<pre><code>
<!DOCTYPE html>
<html>
<head>

  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Server Monitor Demo</title>
  <script src="http://127.0.0.1:5413/sgcWebSockets.js"></script>
  <script src="http://127.0.0.1:5413/esegece.com.js"></script>
  <link rel="stylesheet" href="http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.css" />
  <script src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
  <script src="http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.js"></script>
  <script type="text/javascript" src='https://www.google.com/jsapi'></script>
  <style>
    #status {
      padding: 5px;
      color: #fff;
      background: #ccc;
    }
    #status.fail {
      background: #c00;
    }
    #status.offline {
      background: #c00;
    }
    #status.online {
      background: #0c0;
    }
  </style>
  <script type="text/javascript">
    var vMemory;
    var vCpu;
    var vNetwork;
    var chart;
    var data;
    var options;
    var ws;

    vMemory=30;
    vCpu=55;
    vNetwork=68;
    google.load('visualization', '1', {packages:['gauge']});
    google.setOnLoadCallback(drawChart);
    function drawChart() {
      data = google.visualization.arrayToDataTable([
        ['Label', 'Value'],
        ['Memory', vMemory],
        ['CPU', vCpu],
        ['Network', vNetwork]
      ]);
      options = {
        width: 400, height: 120,
        redFrom: 90, redTo: 100,
        yellowFrom:75, yellowTo: 90,
        minorTicks: 5
      };
      chart = new google.visualization.Gauge(document.getElementById('chart_div'));
      chart.draw(data, options);
    }

    function updateChart() {
```

```

        data = google.visualization.arrayToDataTable([
            ['Label', 'Value'],
            ['Memory', vMemory],
            ['CPU', vCpu],
            ['Network', vNetwork]
        ]);
        chart.draw(data, options);
    }

    function subscribe(Channel)
    {
        if (document.getElementById(Channel).checked) {
            ws.subscribe(Channel);
        } else {
            ws.unsubscribe(Channel);
        }
    }

    function wsmonitor()
    {
        if ("WebSocket" in window)
        {
            ws = new SGCWS("ws://127.0.0.1:5413");
            ws.on('open', function(evt){
                document.getElementById('status').innerHTML = "Socket Open";
                document.getElementById('status').className = "online";
                ws.subscribe("memory");
                ws.subscribe("cpu");
                ws.subscribe("network");
            });
            ws.on('close', function(evt){
                document.getElementById('status').innerHTML = "Socket Closed";
                document.getElementById('status').className = "offline";
            });
            ws.on('sgcevent', function(evt){
                if (evt.channel == "memory") {
                    vMemory = parseInt(evt.message);
                } else if (evt.channel == "cpu") {
                    vCpu = parseInt(evt.message);
                } else if (evt.channel == "network") {
                    vNetwork = parseInt(evt.message);
                }
                updateChart();
            });
            ws.on('error', function(evt){
                document.getElementById('status').innerHTML = "Socket Error";
                document.getElementById('status').className = "fail";
            });
        }
    }
}
</script>
</head>
<body>
<div data-role="page" id="wsdemo_monitor">
    <div data-role="header" data-theme="b">
        <h1>Server Monitor</h1>
        <a href="#home" data-icon="home" data-iconpos="notext" data-direction="reverse" class="ui-btn-left">
    </div><!-- /header -->
    <div data-role="content">
        <h2>Press Start to Get Monitor Data</h2>
        <p id="status" class="success"></p>
        <h4>Select which data you want to receive: Memory - CPU - Network</h4>
        <a href="javascript:wsmonitor()" data-role="button" data-inline="true">Start</a>
        <div id="chart_div"></div>
        <div data-role="fieldcontain">
            <fieldset data-role="controlgroup" data-type="horizontal">
                <input type="checkbox" name="memory" id="memory" class="custom" checked="True" onclick=
                <label for="memory">Memory</label>
                <input type="checkbox" name="cpu" id="cpu" class="custom" checked="True" onclick=
                <label for="cpu">CPU</label>
                <input type="checkbox" name="network" id="network" class="custom" checked="True"
                <label for="network">Network</label>
            </fieldset>
        </div>
    </div><!-- /content -->
    <div data-role="footer" class="footer-docs" data-theme="c">
        <p>&copy; 2020 eSeGeCe.com</p>
    </div>
</div><!-- /page -->
</body>
</html>
</code></pre>

```


デモ | サーバースナップショット

このデモでは、サーバーからクライアントに画像を送信する方法と、サーバーコンポーネントのブロードキャストメソッドを使用してすべてのクライアントが同じ画像を受信する方法を示します。

設定

- まず新しい [TsgcWebSocketHTTPServer](#) インスタンスを作成し、[サーバー チャット デモ](#)を参照してください。
- メッセージをクライアントに送信する際に送信バイト数を減らすために圧縮を有効にします

```
Server.Extensions.PerMessage_Deflate.Enabled = true
```

- その後、サーバーは 5 秒ごとにすべての接続クライアントに画像ストリームをブロードキャストします。

```
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

このデモでは [TsgcWebSocketClient](#) を使用してバイナリ WebSocket メッセージを読み取る方法を示します。WebSocket サーバーに接続し、ユーザーに表示される画像であるストリームを受信します。

サーバーへ接続

- まず [TsgcWebSocketClient](#) の新しいインスタンスを作成します。
- 次にサーバーの **Host** と **Port** を設定します。
- サーバーからメッセージが送信される際に受信バイト数を減らすために圧縮を有効にします。

```
Client.Extensions.PerMessage_Deflate.Enabled = true
```

- サーバーが送信した画像はストリームとして届くため、画像を読み取るには **OnBinary** イベントを使用します。

```
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);
    }
}
```

デモ | ファイルのアップロード

このデモは、WebSocketプロトコルを使用してWebブラウザからサーバーにファイルをアップロードする方法を示します。

設定

- まず新しい [TsgcWebSocketServer](#) インスタンスを作成し、[サーバーチャットデモ](#)を確認してください。
- ファイルはバイナリストリームとしてサーバーに届きます。ファイルを読み取るには **OnBinary** イベントを処理する必要があります。

```
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);
}
```

- ファイルの名前を知りたい場合は、ファイルが送信される前にファイル名を含むテキストメッセージを送信できます

```
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);
}
```

JavaScript コードを使用して **WebSocket** でファイルを送信する方法を以下に示します。

```
<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>
```

デモ | サーバー認証

このデモはサーバー認証の使用方法を示しています。異なる認証の種類について詳しく知りたい場合は、[認証](#)に関する次の記事をお読みください。

認証

- まず `TsgcWebSocketServer` の新しいインスタンスを作成します。Authentication プロパティを有効にします。 `server.Authentication.Enabled = true;`
- 次に、**OnAuthentication** イベントハンドラーでユーザー名とパスワードが正しいか確認します。正しい場合は `Authenticated` プロパティを `true` に設定し、そうでない場合は `false` に設定します。

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

デモ | KendoUI_Grid

このデモは、KendoUI Grid が WebSocket をプロトコルとして使用し、Web ブラウザーをクライアントとして動作させる方法を示しています。基本的には、クライアントのいずれかが変更を行うと更新される JavaScript グリッドで、これらの変更は WebSocket プロトコルを使用して接続されているすべてのクライアントに更新されるため、すべてのクライアントがクライアントによる変更を含む同じデータをリアルタイムで確認できます。

設定

- まず新しい [TsgcWebSocketHTTPServer](#) インスタンスを作成し、[サーバー チャット デモ](#)を参照してください。
- ウェブブラウザクライアントからリクエストされた必要なファイルを送信するために OnCommandGet を処理する必要があります。

```
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;
    }
}
```

WebSocket 更新

クライアントがグリッドレコードを更新すると、この変更はwebsocketプロトコルを使用して接続中のすべてのクライアントに送信されます。グリッドの変更について通知を受け取るには、OnMessageイベントを使用してください。メッセージはJSON形式なので、JSONテキストを読み取り、デコードして、他のピアに応答を送信するだけです。

```
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++)
            {
                // ...
            }
        }
    }
    catch (...)
    {
        // ...
    }
}
```

```

{
    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();
}
}

```

デモ | ServerSentEvents

このデモは、WebSocket ServerでServer-Sent Eventsがどのように機能するかを示しています。sgcWebSocketsでは、サーバーが同じリッスンポートで複数のプロトコルを処理できます。

[Server Sent Events](#)の詳細については、こちらをご覧ください。

このデモは、サーバーがServer Sent Eventsを使用して、接続中のすべてのクライアントへ毎秒時刻を送信する方法を示します。

サーバーが起動すると、接続しているすべてのクライアントに**サーバー 時刻**を含むメッセージを**ブロードキャスト**します。そのため、クライアントがこのメッセージを受信するたびに、ユーザーに表示されます。

```
private void Timer1Timer(TOobject *sender)
{
    server->Broadcast("data: " + "Server Time: " + FormatDateTime("hh:nn:ss", Now));
}
```

WebSocket 接続を処理する **JavaScript** コードを以下に示します:

```
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";
});
```

Demos | Server WebRTC

このデモでは、[TsgcWebSocketHTTPServer](#) と JavaScript ライブラリとして WebRTC を使用してビデオ会議サーバーを構築する方法を示します。

このデモは WebSocket プロトコルを使用して WebRTC をシグナリングし、公開 STUN/TURN サーバーを使用しています。本番サイトでは独自の STUN/TURN サーバーを使用する必要があります。登録済みユーザーは Windows 向けの Coturn をダウンロードできます。すでにコンパイルされており、サーバーで実行するために必要なすべてのライブラリが含まれています。

クライアントは WebRTC 接続をサポートするウェブブラウザでなければなりません。

設定

- まず新しい [TsgcWebSocketHTTPServer](#) インスタンスを作成し、[サーバー チャット デモ](#)を参照してください。
- 次に、[TsgcWSPServer_WebRTC](#)の新しいインスタンスを作成します。
- 接続が開始されたら、WebRTC プロトコルを WebSocket サーバーに割り当て、サーバーのホストとポートを設定する必要があります。

```
WSServer->Port = StrToInt(txtDefaultPort->Text);  
// ... bindings  
With WSServer->Bindings->Add do  
{  
    Port = StrToInt(txtDefaultPort->Text);  
    IP = txtHost>Text;  
}  
// ... active  
WSServer->Active = true;
```

このデモには、WebRTC フロントページの配信に使用するインデックス HTML ページが必要です。このページはデモに同梱されています。

ウェブブラウザで実行

サーバーを設定したら、起動して利用可能なウェブブラウザのいずれかを選択します。新しいウェブブラウザのセッションが開き、新しいセッションの開始を求めます。成功すると自分のビデオが表示され、同じ URL を別のウェブブラウザで開くと、両方のピアが接続されているのが確認できます。

デモはデフォルトで SSL なしで実行されます。これはローカルホスト接続にのみ有効です。本番サイトでは SSL 接続を使用してください。サーバー側での SSL の設定については [サーバーチャットデモ](#)を確認してください。

デモ | サーバー AppRTC

このデモでは、[TsgcWebSocketHTTPServer](#) と JavaScript ライブラリとして AppRTC を使用してビデオ会議サーバーを構築する方法を示します。

このデモは WebSocket プロトコルを使用して WebRTC をシグナリングし、公開 STUN/TURN サーバーを使用しています。本番サイトでは独自の STUN/TURN サーバーを使用する必要があります。登録済みユーザーは Windows 向けの Coturn をダウンロードできます。すでにコンパイルされており、サーバーで実行するために必要なすべてのライブラリが含まれています。

クライアントは WebRTC 接続をサポートするウェブブラウザでなければなりません。

設定

- まず新しい [TsgcWebSocketHTTPServer](#) インスタンスを作成し、[サーバー チャット デモ](#)を参照してください。
- 次に、[TsgcWSPServer_AppRTC](#) の新しいインスタンスを作成します。
- 次に、AppRTC プロトコルを WebSocket サーバーに割り当て、サーバーのホストとポートを設定する必要があります。WebRTC はセキュアな接続を必要とするため、PEM 証明書を使用してコンポーネントの SSLOptions プロパティを設定する必要があります。

```
WSServer->Port = StrToInt(txtDefaultPort->Text);
// ... bindings
With WSServer->Bindings->Add do
{
    Port = StrToInt(txtDefaultPort->Text);
    IP = txtHost->Text;
}
// ... properties
WSPAppRTC->AppRTC->RoomLink = "https://" + txtHost->Text + ":" + txtDefaultPort->Text + "/r/";
WSPAppRTC->AppRTC->WebSocketURL = "wss://" + txtHost->Text + ":" + txtDefaultPort->Text;
// ... active
WSServer->Active = true;
```

- AppRTC.RoomLink は、ウェブブラウザがルームにログインするためにリダイレクトされる URL です。
- AppRTC.WebSocketURL は WebSocket 接続の URL です。
- IceServers は AppRTC サーバープロトコルで設定できます。

デモには、AppRTCのフロントページをディスパッチするために使用されるインデックスHTMLページが必要です。このページはデモとともに提供されます。

ウェブブラウザで実行

サーバーを設定したら、起動して利用可能な Web ブラウザーのいずれかを選択します。新しいルームに参加するよう求める新しい Web ブラウザーセッションが開きます。このルームに参加し、成功すると別の Web ブラウザーから新しいビデオ会議を開始するために使用するリンクが表示されます。

AppRTC

Please enter a room name.

959454873

JOIN

RANDOM

Recently used rooms:

Demos | Telegram Client

このデモは Telegram への接続、すべての連絡先の受信、テキストメッセージの送信、画像の送信など多くのことを示しています。

設定

- [TsgcTDLib_Telegram](#) の新しいインスタンスを最初に作成してください。
- 次に、Telegram に接続を試みる前に、API Hash、API ID などのパラメータをクライアントコンポーネントに渡す必要があります。必要なすべてのパラメータを設定したら、プロパティ `Active = true` を設定して接続を開始してください。

```
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;
```

- クライアントが Telegram への接続を試みるときに、通常コードが必要です。そのため **OnTelegramAuthenticationCode** を処理して、Telegram アカウントから提供された値で Code パラメータを返す必要があります。

```
private void OnAuthenticationCodeEvent(TObject *Sender, ref string Code)
{
    Code = InputBox("Telegram", "Introduce Telegram Code");
}
```

Telegram メッセージの送信

Telegram メッセージ（テキスト、ファイル、画像など）を送信するには、まずメッセージを送信したい ChatId を設定し、次にテキストメッセージやファイル名などのパラメータを設定する必要があります。

```
// send text message
sgcTelegram->SendTextMessage("456413", "Hello From sgcWebSockets!!!");

// send file message
sgcTelegram->SendDocumentMessage("383784", "c:\yourfile.txt");
```

Telegramメッセージを受信する

Telegram クライアントが受信したメッセージは、特定のイベントハンドラーで処理されます。次のテキストメッセージが受信されたとき、新しいドキュメントが受信されたとき、写真など、それぞれにイベントがあります。

```
private void OnMessageTextEvent(TObject *Sender, TsgcTelegramMessageText *MessageText)
{
```

```
DoLogMessage(MessageText->ChatId, MessageText->SenderId->ToString(), MessageText->Text);
}

private void OnMessageDocumentEvent(TsgcTDLib_Telegram *Sender, TsgcTelegramMessageDocument *MessageDocument)
{
    DoLogMessage(MessageDocument->ChatId, MessageDocument->SenderId->ToString(), MessageDocument->FileName);
}
```

Coturn

Coturn

sgcWebSockets 4.5.2 ENTERPRISE エディション以降、Delphi/CBuilder を使用して独自の STUN/TURN サーバーを構築できます。

TURN および STUN サーバーの無料オープンソース実装です。

TURN サーバーは VoIP メディアトラフィック NAT トラバーサルサーバーおよびゲートウェイです。汎用ネットワークトラフィック TURN サーバーおよびゲートウェイとしても使用できます。

サポートされるプロジェクトのターゲットプラットフォームは次のとおりです:

- **Linux**
- **Mac OS X**
- **Windows (Cygwin):** コンパイル済みバイナリは登録ユーザーが利用できます。

Windows 設定

まず、アカウントからコンパイル済みのバイナリをダウンロードする必要があります。win32とwin64の2つのバージョンが利用可能です。希望するプラットフォームを選択し、バイナリをフォルダーに解凍します。次のファイルが作成されます:

1. アプリケーションの実行に必要な一部のcygwinライブラリがあります。これらのライブラリはcoturnサーバーとともにデプロイする必要があります。
2. いくつかのコンソールアプリケーション:
 - 2.1 turnserver.exe: TURN/STUN サーバーを実行するメインコンソールアプリケーションです。
 - 2.2 その他のアプリケーション: 設定またはテスト目的に使用します。
3. Turnserver.conf: coturnサーバーの設定ファイルです。

 cygcrypto-1.0.0.dll	 turnserver.conf
 cygcrypto-1.1.dll	 turnserver.exe
 cygevent_core-2-0-5.dll	 turnutils_natdiscovery.exe
 cygevent_extra-2-0-5.dll	 turnutils_oauth.exe
 cygevent_openssl-2-0-5.dll	 turnutils_peer.exe
 cygevent_pthreads-2-0-5.dll	 turnutils_rfc5769check.exe
 cyggcc_s-1.dll	 turnutils_stunclient.exe
 cygiconv-2.dll	 turnutils_uclient.exe
 cygintl-8.dll	
 cyglber-2-4-2.dll	
 cyglldap_r-2-4-2.dll	
 cygmariadb-3.dll	
 cygpq-5.dll	
 cygsasl2-3.dll	
 cygssl-1.0.0.dll	
 cygssl-1.1.dll	
 cygwin1.dll	
 cygz.dll	

turnserver.conf

これは coturn サーバーの設定ファイルです。開くとデフォルト設定が表示されます。

シンプルな設定

サーバーのパブリック IP アドレスは 80.15.44.123 で、ポート 80 でリスニングしています。接続の資格情報は以下のとおりです: ユーザー名 = demo、パスワード = secret
以下の設定を行います:

```
listening-ip=80.15.44.123
listening-port=80
realm=yourrealm.com
user=demo:secret
```

TLS を有効にした設定

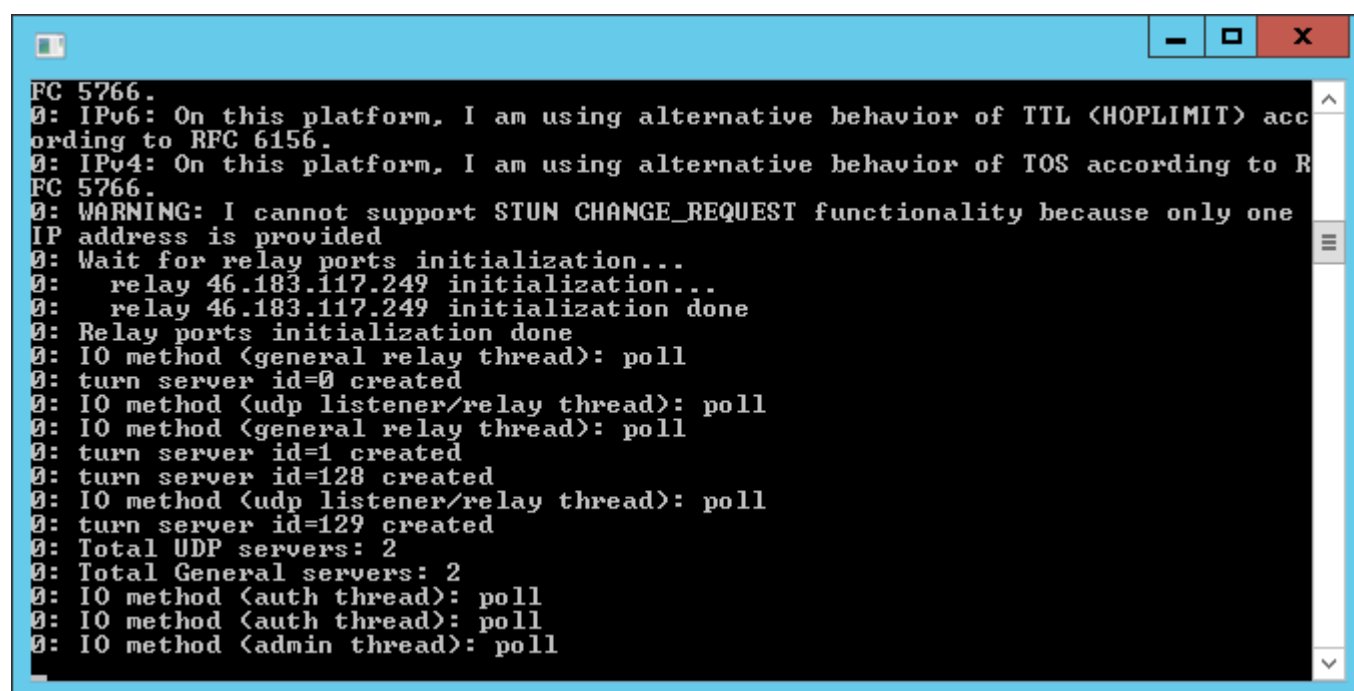
サーバーのパブリック IP は 80.15.44.123 でポート 80 および 443 (TLS 接続) でリスンしています。接続の認証情報は: ユーザー名 = demo、パスワード = secret。証明書名 (PEM 形式である必要があります) は certificate.crt で、秘密鍵は private.key です。
以下の設定を行います:

```
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
```

利用可能な設定は他にもあります。turnserver.confを開いて、ドキュメント化されたセクションを読んでください。

Run coturn

設定が完了すると、turnserver.exe を実行するだけでサーバーを起動でき、新しいコンソールアプリケーションが開いてログファイルが作成されます。turnserver.conf ファイルで "verbose" を有効にすると、コンソールアプリケーションの詳細レベルを上げる (より詳細なメッセージを取得する) ことができます。



```
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
```


ライセンス

eSeGeCe コンポーネントエンドユーザーライセンス契約

eSeGeCe コンポーネント（「eSeGeCe」）使用許諾契約書（「EULA」）は、eSeGeCeのすべてのコンポーネント（関連するソフトウェアコンポーネント、メディア、印刷物、および「オンライン」または電子ドキュメント（「eSeGeCe コンポーネント」）を含む場合があります）について、お客様（個人または単一の法人）とeSeGeCeの著者との間で締結される法的契約です。eSeGeCe コンポーネントをインストール、コピー、またはその他の方法で使用するにより、お客様はこのEULAの条項に同意したものとみなされます。この使用許諾契約書は、お客様とeSeGeCeの著者（「ライセンサー」と称します）との間のプログラムに関する完全な合意を表し、当事者間の以前のすべての提案、表明、または了解事項に優先します。このEULAの条項に同意しない場合は、eSeGeCe コンポーネントをインストールまたは使用しないでください。

eSeGeCe コンポーネントは著作権法および国際著作権条約、ならびにその他の知的財産法および条約によって保護されています。eSeGeCe コンポーネントはライセンスされるものであり、販売されるものではありません。

ソースコードが必要な場合は登録料を支払う必要があります。eSeGeCe のライセンスキーおよび/またはフルエディション（DCU エディションおよびソースエディションを含む）を第三者の個人および/または法人に提供してはなりません。また、第三者の個人および/または法人から eSeGeCe のライセンスキーおよび/またはフルエディションを使用してはなりません。

1. ライセンスの付与

eSeGeCeコンポーネントは次のようにライセンスされています。

(a) インストールおよび使用。

LICENSER は、eSeGeCe コンポーネントが設計された OS（例：Windows 2000、Windows 2003、Windows XP、Windows ME、Windows Vista、Windows 7、Windows 8、Windows 10）の有効なライセンスコピーが動作しているコンピューターに eSeGeCe コンポーネントのコピーをインストールして使用する権利を付与します。

(b) ロイヤリティなし。

eSeGeCe コンポーネントをベースにした商用アプリケーションを作成して 実行ファイルとともに配布できます。

ロイヤリティは不要です。

(c) 変更（ソースエディションのみ）。

このライセンス契約に定められた条件の下で提供されたライセンス済みソースコードに対して、修正、拡張、派生作品、および/または拡張機能を作成することができます。

(d) バックアップコピー。

バックアップおよびアーカイブ目的で必要の場合は、eSeGeCe コンポーネントのコピーを作成できます。

2. その他の権利および制限事項の説明

(a) 著作権表示の維持。

eSeGeCeコンポーネントのすべてのコピーから、著作権表示を削除または変更してはなりません。

(b) 配布。

eSeGeCe コンポーネントの登録済みコピーを第三者に配布することはできません。eSeGeCe 公式ウェブサイトからダウンロード可能な評価版は自由に配布できます。

アプリケーション用に eSeGeCe コンポーネントを含む コンポーネント/ActiveX コントロール/ライブラリを作成することはできますが、第三者に配布または公開してはなりません。

(c) SOURCE CODE の配布の禁止（Source エディションのみ）。

ソースコード、またはその変更、拡張、派生物や拡張機能をソースコード形式で第三者に配布または公開してはなりません。

ソースコードのいかなる部分も、第三者に配布、公開、開示、またはその他の方法で提供してはなりません。

(d) リバースエンジニアリング、逆コンパイル、逆アセンブルの禁止。

この制限によって適用法が明示的に許可している場合を除き、eSeGeCe コンポーネントのリバースエンジニアリング、逆コンパイル、逆アセンブルを行うことはできません。

(e) レンタル。

eSeGeCe コンポーネントを賃貸、リース、または貸与することはできません。

(f) サポートサービス。

ライセンサーは eSeGeCe コンポーネントに関連するサポートサービス（「サポートサービス」）を提供する場合があります。サポートサービスの一部として提供される補足ソフトウェアコードは eSeGeCe コンポーネントの一部とみなされ、この EULA の利用規約に従います。

eSeGeCe のライセンスは、同時に 1 人の開発者のみが使用できます。テクニカルサポートも特定の 1 人の開発者に対してのみ提供されます。

(g) 適用法への準拠。

eSeGeCe コンポーネントの使用に関するすべての適用法令を遵守する必要があります。

3. 終了

他の権利を損なうことなく、この EULA の条件に違反した場合、LICENSER は本 EULA を終了させる場合があります。その場合、お客様の所有するすべての eSeGeCe コンポーネントのコピーを廃棄する必要があります。

4. 著作権

eSeGeCe コンポーネントおよびそのコピーに関するすべての権利（著作権を含むがこれに限らない）は、ライセンサーまたはそのサプライヤーが所有します。eSeGeCe コンポーネントの使用を通じてアクセスできるコンテンツに関するすべての権利および知的財産権は、それぞれのコンテンツ所有者の財産であり、適用される著作権またはその他の知的財産法および条約によって保護される場合があります。この EULA はかかるコンテンツを使用する権利を付与しません。明示的に付与されていないすべての権利はライセンサーが留保します。

5. 保証なし

LICENSERはeSeGeCeコンポーネントに対するいかなる保証も明示的に否認します。eSeGeCeコンポーネントは「現状のまま」提供され、商品性、非侵害性、特定目的への適合性に関する保証を含む（ただしこれらに限らない）いかなる明示的または黙示的な保証也没有。LICENSERは、eSeGeCeコンポーネントに含まれる情報、テキスト、グラフィック、リンクまたはその他の項目の正確性または完全性について保証または責任を負いません。LICENSERは、コンピューターウイルス、ワーム、タイムボム、ロジックボム、またはその他のコンピュータープログラムの送信によって生じる損害に関していかなる保証もしません。LICENSERはさらに、承認されたユーザーまたは第三者に対するいかなる保証または表明も明示的に否認します。

6. 責任の制限

ライセンサーは、ライセンサーがかかる損害の可能性について事前に告知を受けていた場合であっても、「承認されたユーザー」によるeSeGeCeコンポーネントの使用または使用不能から生じるいかなる損害（逸失利益、事業の中断、または情報の喪失を含むがこれらに限定されない）についても責任を負いません。いかなる場合においても、ライセンサーはデータの喪失、または間接的、特別、付随的、結果的（逸失利益を含む）、またはその他の契約上、不法行為上、またはその他に基づく損害について責任を負いません。ライセンサーは、eSeGeCe コンポーネントの内容またはその一部（コンポーネントに含まれるエラーや省略、名誉毀損、パブリシティ権の侵害、プライバシー、商標権、事業の中断、人身傷害、プライバシーの喪失、精神的権利、または秘密情報の開示を含むがこれらに限定されない）に関していかなる責任も負いません。

インデックス

- AbstractApi Geolocation [1113](#)
- Add Telegram Proxy [671](#)
- ALPN [128](#)
- Amazon SQS [957](#)
- AMQP Channels [316](#)
- AMQP Consume Messages [324](#)
- AMQP Exchanges [318](#)
- AMQP QoS [327](#)
- AMQP トランザクション [328](#)
- AMQP パブリッシュメッセージ [323](#)
- AMQP メッセージの取得 [326](#)
- AMQP1 Links [354](#)
- AMQP1 Send Message [360](#)
- AMQP1 セッション [352](#)
- AMQP1 メッセージ読み取り [362](#)
- AMQP1 レシーバーリンク [358](#)
- AMQP1 送信者リンク [355](#)
- Anthropic [760](#)
- Anthropic Citations [779](#)
- Anthropic Documents [775](#)
- Anthropic MCP コネクタ [786](#)
- Anthropic Structured Outputs [782](#)
- Anthropic Tool Use [769](#)
- Anthropic ウェブ検索 [780](#)
- Anthropic バッチ [772](#)
- Anthropic ビジョン [767](#)
- Anthropic ファイル [784](#)
- Anthropic プロンプトキャッシング [777](#)
- Anthropic メッセージ [765](#)
- Anthropic モデル [771](#)
- Anthropic 拡張思考 [773](#)
- API [424](#), [448](#), [449](#), [456](#), [472](#), [484](#), [506](#), [604](#), [1112](#)
- API 3Commas [560](#)
- API Binance [426](#), [442](#), [448](#)
- API Binance Futures [442](#), [448](#)
- API Binance Futures 取引 [448](#)
- API Bitfinex [528](#)
- API Bitget [617](#)
- API Bitmex [520](#)
- API Blockchain [590](#)
- API Bybit [575](#)
- API Cex [592](#)
- API Cex Plus [599](#)
- API Crypto.com [625](#)
- API Deribit [622](#)
- API Discord [604](#)
- API Forex [580](#)
- API GateIO [620](#)
- API HTX [628](#)
- API Kraken [472](#), [484](#), [486](#), [491](#), [498](#), [506](#)
- API Kraken Futures [491](#), [498](#), [506](#)
- API Kraken Futures REST プライベート [506](#)
- API Kraken REST Private [486](#)
- API Kraken REST パブリック [484](#)
- API Kucoin [532](#)
- API Kucoin 先物 [546](#)
- API MEXC [609](#)
- API MEXC フューチャーズ、API [613](#)
- API OKX [566](#)
- API OpenAI [607](#)
- API Pusher [511](#)
- API SignalR [469](#)
- API SignalRCore [463](#)
- API SocketIO [449](#)
- API Whatsapp [630](#)
- API XTB [572](#)
- API、API Coinbase、API Coinbase Pro [451](#)
- API、API Kraken、API Kraken Futures REST
パブリック、API Kraken Futures [504](#)
- API、API Telegram [655](#)
- API、WebSocket、API Kraken、WebSocket
プライベート [481](#)
- API、WebSockets、Forex Connect、
Lightstreamer [587](#)
- APIs [426](#), [434](#), [437](#), [442](#), [459](#), [463](#), [469](#), [486](#),
[491](#), [498](#), [511](#), [520](#), [528](#), [560](#), [580](#), [590](#), [592](#),
[609](#)
- APIs、API Kraken、API Kraken Futures [489](#)
- APIs、WebSockets、API Kraken、
WebSockets パブリック [474](#)
- APNs [861](#)

- Certificate-Based Connection [861](#)
 - APNs [860](#)
 - 通知リクエスト [859](#)
- Apple Push Notifications [857](#)
- Authentication [166](#), [191](#)
- Authorization Code Grant (RFC 6749) [876](#)
- Binance Connect [434](#)
- Binance Private REST API [437](#)
- Binance Subscribe [435](#)
- Binance スポット取引 [438](#)
- Binance プライベートリクエストの時刻 [440](#)
- Binance マーケットデータの取得 [436](#)
- Binance 出金 [441](#)
- Binary Message [163](#)
- Bindings [178](#)
- Bitmex Subscribe WebSocket Channel [525](#)
- Bitmex WebSocket API への接続 [524](#)
- Bitmex注文の発注方法 [526](#)
- Broadcast [117](#)
- Build [75](#), [76](#)
- Build Android Application [75](#)
- Certificate-Based Connection [861](#)
 - APNs [861](#)
- Certificates OpenSSL [158](#)
- Channels [117](#), [316](#), [435](#), [1028](#)
- Client [163](#), [166](#), [169](#), [171](#), [1016](#), [1025](#), [1027](#), [1028](#), [1118](#)
- Client Authentication [166](#)
- Client Credentials Grant (RFC 6749) [880](#)
- Client Exceptions [169](#)
- Client Register Protocol [171](#)
- Client Send Text [163](#)
- Clients [1118](#)
- Coinbase Pro Connect [456](#)
- Coinbase Pro Place Orders [461](#)
- Coinbase Pro Private REST API [459](#)
- Coinbase Pro サンドボックスアカウント [462](#)
- Coinbase Pro プライベートリクエスト時刻 [460](#)
- Coinbase Pro マーケットデータ取得 [458](#)
- CommitOffset [339](#)
- CommitSync [337](#)
- Configure Install [55](#)
- Configure ZLib [63](#)
- Connect WebSocket Server [148](#)
- Connections TIME_WAIT [155](#)
- Coturn [1137](#)
- Cross-Site Scripting [240](#)
- CryptoHopper [674](#)
- Custom Sub [108](#)
- Datasnap [1046](#)
- DeepSeek [791](#)
- DeepSeek Vision [794](#)
- DeepSeek メッセージ [793](#)
- DeepSeek モデル [795](#)
- Deflate-Frame [686](#)
- DLL の使用 [105](#)
- DNSBL [246](#)
- DPoP - Demonstrating Proof of Possession (RFC 9449) [886](#)
- DPoP 検証 (RFC 9449) [917](#)
- EPOLL [127](#)
- Extensions [684](#)
- FCrDNS [246](#)
- Files [119](#), [200](#), [412](#)
- Fired [227](#)
- Firewall ブラックリスト、Firewall ホワイトリスト、IPフィルタリング、CIDR、TsgcWebSocketFirewall [236](#)
- Flash [122](#)
- Forex Connect [582](#)
- Forex Market Data [586](#)
- Forex Trade [583](#)
- Forex 口座 [585](#)
- Gemini [788](#)
- Gemini Embeddings、Gemini EmbedContent [804](#)
- Gemini Vision [800](#)
- Gemini ツール使用、Gemini ファンクション呼び出し [805](#)
- Gemini トークンカウント、Gemini CountTokens [803](#)
- Gemini メッセージ [798](#)
- Gemini モデル [801](#)
- Gemini 構造化出力 [802](#)
- Generate [858](#)
 - Remote Notification APNs [858](#)
- GetCommittedOffset [339](#)
- GetEarliestOffset [339](#)
- GetLatestOffset [339](#)
- Google Calendar [981](#), [987](#), [988](#), [989](#)
- Google Calendar RefreshToken [989](#)

- Google Calendar サービスアカウント [990](#)
- Google Calendar 同期 カレンダー [987](#)
- Google Calendar 同期イベント [988](#)
- Google Cloud FCM [991](#)
- Google Cloud Pub [972](#)
- Google OAuth2 Keys [962](#)
- Google Service Accounts [968](#)
- Grok [813](#)
- Grok Vision [816](#)
- Grok メッセージ [815](#)
- Grok モデル [817](#)
- GroupId [337](#)
- HeartBeat [113](#)
- HighPerfAcceptsPerWorker [218](#)
- HTTP [68](#), [116](#), [129](#), [199](#), [200](#), [207](#), [208](#), [839](#), [843](#)
- HTTP Dispatch Files [200](#)
- HTTP リクエストの転送 [129](#)
- HTTP、HTTPAPI、HTTPAPI HTTP の無効化 [222](#)
- HTTP/2 [201](#), [202](#), [204](#), [844](#), [845](#), [853](#)
- HTTP/2 Server Push [844](#)
- HTTP/2 サーバースレッド [205](#)
- HTTP/2 サーバプッシュ [202](#)
- HTTP/2 ファイルダウンロード [845](#)
- HTTP/2 ヘッダー、HTTP/2 [847](#)
- HTTP/2 代替サービス [204](#)
- HTTP/2 切断理由、HTTP/2 [850](#)
- HTTP/2 部分レスポンス、HTTP/2 [846](#)
- HTTP1 [863](#)
- HTTP2 [840](#)
- HTTPAPI [220](#), [227](#)
- HTTPAPI FineTune [218](#)
- HTTPAPI OnDisconnect [227](#)
- HTTPAPI Send File Response [225](#)
- HTTPAPI URL Reservation [216](#)
- HTTPAPI テキストレスポンスの送信 [224](#)
- HTTPAPI、HTTPAPI カスタムヘッダー [223](#)
- HTTPサーバーストリームビデオ [210](#)
- ICE [1033](#)
- ICE ペア候補 [1036](#)
- ICE候補の収集 [1035](#)
- Indy [101](#)
- Indy SChannel [183](#), [211](#)
- Install Errors [51](#)
- Install Setup [33](#)
- IOCP [126](#)
- iOS アプリケーションのビルド [76](#)
- IoT [825](#), [834](#)
- IoT Azure MQTT クライアント [834](#)
- JWT [926](#)
- Kafka [334](#), [335](#), [336](#), [337](#), [339](#)
- Kafka オフセット [339](#)
- Kafka コンシューマーグループ [337](#)
- Kafka メッセージのプロデュース [335](#)
- Kafka メッセージの消費 [336](#)
- Kafka、Kafka トピック、CreateTopic、DeleteTopic、GetMetadata [338](#)
- KendoUI_Grid [1129](#)
- Kucoin Futures Private Requests Time [559](#)
- Kucoin Futures Subscribe WebSocket Channel [553](#)
- Kucoin Futures WebSocket API への接続 [552](#)
- Kucoin Futures プライベート REST API [555](#)
- Kucoin Futures 取引 [556](#)
- Kucoin Futures 市場データの取得 [554](#)
- Kucoin Private REST API [542](#)
- Kucoin Subscribe WebSocket Channel [540](#)
- Kucoin Trade Spot [543](#)
- Kucoin WebSocket API への接続 [539](#)
- Kucoin プライベートリクエストの時間 [545](#)
- Kucoin 市場データの取得 [541](#)
- LoadBalancing [142](#)
- MaxConnectionsPerIP [243](#)
- MaxMessagesPerSec [243](#)
- MCP [687](#)
- MCP Server Roots [700](#)
- MCP Server Sessions [690](#)
- MCP Server Tools [691](#)
- MCP クライアントサンプリング [713](#)
- MCP クライアントツール [706](#)
- MCP クライアントプロンプト [708](#)
- MCP クライアントリソース [710](#)
- MCP クライアントルーツ [712](#)
- MCP サーバーエリシテーション [703](#)
- MCP サーバーサンプリング [701](#)
- MCP サーバープロンプト [694](#)
- MCP サーバーリソース [697](#)
- MCPクライアント エリシテーション [714](#)
- Message Filtering [240](#)
- Method [843](#)
- Mistral AI [818](#)

- Mistral ビジョン [822](#)
- Mistral メッセージ [821](#)
- Mistral モデル [823](#)
- Mistral 埋め込み [824](#)
- MQTT [298](#), [300](#), [305](#), [307](#), [309](#), [834](#), [1118](#)
- MQTT Receive Messages [307](#)
- MQTT サブスクライブ [305](#)
- MQTT 保持メッセージのクリア [309](#)
- MQTT、IoT、IoT Amazon MQTT クライアント [826](#)
- MQTT、MQTT トピック [303](#)
- MQTT、MQTT パブリッシュ/サブスクライブ、MQTT パブリッシュ [302](#)
- MQTT、MQTT パブリッシュメッセージ、MQTT パブリッシュ [306](#)
- MQTT、クライアント MQTT バージョン、クライアント [301](#)
- MQTT、サーバー、Mosquitto 接続 [299](#)
- MQTT、レスポンス待機、MQTT Publish [308](#)
- OAuth2 [853](#), [867](#), [897](#), [898](#), [899](#), [900](#), [901](#), [902](#), [962](#)
- OAuth2 None Authenticate URLs [902](#)
- OAuth2 Provider Authentication [923](#)
- OAuth2 Register Apps [899](#)
- OAuth2 アクセストークンの回復 [900](#)
- OAuth2 サインインの HTML カスタマイズ [897](#)
- OAuth2 プロバイダー Azure AD [921](#)
- OAuth2 プロバイダーのプライベートエンドポイント [922](#)
- OAuth2 プロバイダーリクエスト [925](#)
- OAuth2、サーバー、サーバー例 [893](#)
- Ollama [807](#)
- Ollama Models [811](#)
- Ollama メッセージ [810](#)
- Ollama 埋め込み [812](#)
- ompClassic [218](#)
- ompHighPerf [218](#)
- OpenAI [715](#)
- OpenAI Audio [726](#), [743](#)
- OpenAI Batch [732](#)
- OpenAI Completion [723](#)
- OpenAI Edit [725](#)
- OpenAI RealTime [728](#)
- OpenAI Speech [730](#)
- OpenAI アップロード [733](#)
- OpenAI アプリケーション [734](#)
- OpenAI チャット [724](#)
- OpenAI ファインチューニング [731](#)
- OpenAI モデレーション [727](#)
- OpenAI レスポンス [729](#)
- OpenAPI [1051](#)
- OpenAPI Amazon AWS [1072](#)
- OpenAPI Amazon AWS S3 [1080](#)
- OpenAPI Amazon AWS 認証情報 [1078](#)
- OpenAPI API [1064](#)
- OpenAPI Google Cloud [1081](#)
- OpenAPI Google Cloud Calendar [1095](#)
- OpenAPI Google Cloud OAuth2 [1086](#)
- OpenAPI Google Cloud PubSub [1094](#)
- OpenAPI Google Cloud サービスアカウント [1089](#)
- OpenAPI Google Drive [1096](#)
- OpenAPI Microsoft [1098](#)
- OpenAPI Microsoft Graph [1111](#)
- OpenAPI Microsoft OAuth2 コード [1107](#)
- OpenAPI Microsoft OAuth2 資格情報 [1109](#)
- OpenAPI Microsoft アプリケーションの登録 [1104](#)
- OpenAPI Microsoft テナント [1103](#)
- OpenAPI クライアント [1065](#)
- OpenAPI コマンドライン [1060](#)
- OpenAPI サーバー [1068](#)
- OpenAPI サーバー コードファースト [1071](#)
- OpenAPI サーバースペックファースト [1070](#)
- OpenAPI パーサー Pascal [1053](#)
- OpenAPI ライブラリ [1062](#)
- OpenAPI 追加プロパティ [1058](#)
- OpenSSL [85](#), [88](#), [90](#), [92](#), [93](#), [158](#)
- OpenSSL Android [92](#)
- OpenSSL iOS [93](#)
- OpenSSL OSX [90](#)
- OpenSSL P12 証明書 [97](#)
- OpenSSL Windows [88](#)
- OpenSSL 独自 CA 証明書 [95](#)
- OpenSSL 証明書の検証 [98](#)
- OpenSSL 追加関数の読み込み [99](#)
- OperatingMode [218](#)
- Orders [583](#)
- PDF-Back-Cover [1151](#)
- PDF-Front-Cover [1](#)
- PerMessage-Deflate [685](#)

- Pinecone [757](#)
- PKCE を使用した認可コード (RFC 7636) [878](#)
- Poll [336](#)
- Post Big Files [119](#)
- Price Bars [586](#)
- Price Ticks [586](#)
- Produce [335](#)
- Producer [335](#)
- Protocol AMQP [310](#)
- Protocol AMQP1 [340](#)
- Protocol AppRTC [367](#)
- Protocol Dataset [405](#)
- Protocol Dataset Notify Updates [405](#)
- Protocol Default [393](#)
- Protocol Default Javascript [393](#)
- Protocol Kafka [330](#)
- Protocol MQTT [295](#)
- Protocol Presence [413](#)
- Protocol WAMP [372](#)
- Protocol WAMP2 [384](#)
- Protocols [108](#), [171](#), [290](#), [295](#), [310](#), [330](#), [367](#), [372](#), [405](#), [413](#)
- Proxy [671](#)
- QueueLength [218](#)
- QuickStart HTTP [68](#)
- QuickStart WebSockets [66](#)
- RCN [673](#)
- Receive Binary Messages [195](#)
- Refresh Token グラント [909](#)
- Register [672](#)
- Remote Notification APNs [858](#)
 - Generate [858](#)
- Request HTTP [843](#)
- Resource Owner Password Credentials グラント (RFC 6749) [882](#)
- REST API [582](#)
- RTCMultiConnection [679](#)
- RTCPeerConnection DTLS [1044](#)
- RTCPeerConnection ICE [1043](#)
- RTCPeerConnection STUN TURN [1041](#)
- RTCPeerConnection WebSocket クライアント [1040](#)
- RTCPeerConnection WebSocket サーバー [1039](#)
- RTCPeerConnection シグナリング [1042](#)
- RTCPeerConnection データ [1045](#)
- SChannel [183](#), [211](#)
- SChannel Server [183](#), [211](#)
- SChannel 接続情報の取得 [160](#)
- Send Big Files [412](#)
- Send Telegram Invoice Message [669](#)
- Server [177](#), [178](#), [180](#), [187](#), [191](#), [195](#), [220](#), [901](#), [1128](#), [1132](#), [1133](#)
- Server AppRTC [1133](#)
- Server Authentication [191](#), [901](#), [1128](#)
- Server Bindings [178](#)
- Server Keep Active [180](#)
- Server Keep Connections Alive [187](#)
- Server SSL [220](#)
- Server SSL SChannel [183](#), [211](#)
- Server Start [177](#)
- Server-Sent Events [140](#)
- ServerSentEvents [1131](#)
- Service [968](#)
- sgcIndy パッケージのインストール [57](#)
- SkipIOCPOnSuccess [218](#)
- SQL Injection [240](#)
- Stop Limit [583](#)
- STUN [1013](#), [1016](#), [1020](#), [1021](#)
- STUN Client UDP Retransmissions [1016](#)
- STUN サーバー 長期認証情報 [1020](#)
- STUN サーバー代替サーバー [1021](#)
- Sub [972](#)
- SubProtocol [138](#)
- TCP サーバーへの接続 [154](#)
- TCP 接続 [137](#)
- Telegram スポンサーメッセージ [668](#)
- Telegram チャット [667](#)
- Telegram 太字メッセージの送信 [666](#)
- Telegramでスーパーグループメンバーを取得する [670](#)
- Telegramユーザーを登録 [672](#)
- Token Introspection (RFC 7662) [915](#)
- Token Revocation (RFC 7009) [913](#)
- Transactions [135](#)
- TsgcAIChat [749](#)
- TsgcAIDatabaseVectorFile [753](#)
- TsgcAIDatabaseVectorPinecone [754](#)
- TsgcAIOpenAIAssistant [735](#)
- TsgcAIOpenAIAssistant ストリーミング [739](#)
- TsgcAIOpenAIAssistant ファイル検索 [737](#)

[TsgcAIOpenAIAssistant 関数呼び出し](#) 741
[TsgcAIOpenAIChatBot](#) 750
[TsgcAIOpenAIEmbeddings](#) 752
[TsgcAIOpenAITranslator](#) 751
[TsgcAudioPlayerMCI](#) 745
[TsgcAudioRecorderMCI](#) 744
[TsgcBotClassification](#) 246
[TsgcHTTP_API_Anthropic](#) 764
[TsgcHTTP_API_DeepSeek](#) 792
[TsgcHTTP_API_Gemini](#) 790
[TsgcHTTP_API_Grok](#) 814
[TsgcHTTP_API_Mistral](#) 820
[TsgcHTTP_API_Ollama](#) 809
[TsgcHTTP_API_OpenAI](#) 722
[TsgcHTTP_JWT_Client](#) 928
[TsgcHTTP_JWT_Server](#) 931
[TsgcHTTP_OAuth2_Client](#) 868
[TsgcHTTP_OAuth2_Client_Google](#) 874
[TsgcHTTP_OAuth2_Client_Microsoft](#) 875
[TsgcHTTP_OAuth2_Server](#) 890
[TsgcHTTP_OAuth2_Server_Provider](#) 919
[TsgcHTTP2Client](#) 841
[TsgcHTTP2ConnectionClient](#) 854
[TsgcHTTP2RequestProperty](#) 855
[TsgcHTTP2ResponseProperty](#) 856
[TsgcICEClient](#) 1034
[TsgcIWebsocketClient](#) 286
[TsgcIWWSPClient_Dataset](#) 400
[TsgcIWWSPClient_sgc](#) 392
[TsgcRTCPeerConnection](#) 1037
[TsgcServerHTTPAPI_FineTune](#) 218
[TsgcSTUNClient](#) 1014
[TsgcSTUNServer](#) 1019
[TsgcTCPClient](#) 173
[TsgcTextToSpeechAmazon](#) 748
[TsgcTextToSpeechGoogle](#) 747
[TsgcTextToSpeechSystem](#) 746
[TsgcTURNClient](#) 1023
[TsgcTURNServer](#) 1029
[TsgcUDPClient](#) 1011
[TsgcUDPServer](#) 1012
[TsgcWebPush_Client](#) 683
[TsgcWebsocketClient](#) 146
[TsgcWebsocketClient_WinHTTP](#) 228
[TsgcWebsocketFirewall](#) 229, 240, 243, 246
[TsgcWebsocketHTTPServer](#) 197
[TsgcWebsocketHTTPServer_Sessions](#) 208
[TsgcWebsocketLoadBalancerServer](#) 284
[TsgcWebsocketProxyServer](#) 285
[TsgcWebsocketServer](#) 175
[TsgcWebsocketServer_HTTPAPI](#) 214
[TsgcWebView2](#) 994
[TsgcWSAPIClient_MCP](#) 705
[TsgcWSAPIKeyManager](#) 272
[TsgcWSAPIServer_MCP](#) 688
[TsgcWSAPIServer_WebAuthn](#) 933
[TsgcWSAPIServer_WebPush](#) 682
[TsgcWSCircuitBreaker](#) 261
[TsgcWSConnection](#) 288
[TsgcWSHTTP2WebBrokerBridgeServer](#) 1049
[TsgcWSHTTPWebBrokerBridgeServer](#) 1047
[TsgcWSMessageFile](#) 409
[TsgcWSPClient_AMQP](#) 311
[TsgcWSPClient_AMQP1](#) 342
[TsgcWSPClient_Dataset](#) 399
[TsgcWSPClient_E2EE](#) 423
[TsgcWSPClient_Files](#) 408
[TsgcWSPClient_Kafka](#) 331
[TsgcWSPClient_MQTT](#) 297
[TsgcWSPClient_Presence](#) 416
[TsgcWSPClient_sgc](#) 391
[TsgcWSPClient_STOMP](#) 364
[TsgcWSPClient_STOMP_ActiveMQ](#) 366
[TsgcWSPClient_STOMP_RabbitMQ](#) 365
[TsgcWSPClient_WAMP](#) 375
[TsgcWSPClient_WAMP2](#) 385
[TsgcWSPPresenceMessage](#) 415
[TsgcWSPServer_AppRTC](#) 368
[TsgcWSPServer_Dataset](#) 398
[TsgcWSPServer_E2EE](#) 422
[TsgcWSPServer_Files](#) 407
[TsgcWSPServer_Presence](#) 414
[TsgcWSPServer_sgc](#) 390
[TsgcWSPServer_WAMP](#) 374
[TsgcWSPServer_WebRTC](#) 370
[TsgcWSRateLimiter](#) 249
[TsgcWSServer_HTTPAPI_WebBrokerBridge](#) 1050
[TURN](#) 1022, 1025, 1026, 1027, 1028, 1031, 1032
[TURN Client Allocate IP Address](#) 1025
[TURN Client Send Indication](#) 1027

- TURN クライアントパーミッションの作成 [1026](#)
- TURN サーバー 長期認証情報 [1031](#)
- TURN サーバーアロケーション [1032](#)
- WAMP [379](#), [380](#), [382](#)
- WAMP Publishers [380](#)
- WAMP RPC プログレス結果 [382](#)
- WAMP Subscribers [379](#)
- WAMP、WAMP シンプルRPC [381](#)
- WatchDog [114](#)
- Web Browser Test [107](#)
- Web アプリケーション用 OAuth2 クライアント [872](#)
- WebAuthn [932](#)
- WebAuthn Authorization HTTP [951](#)
- WebAuthn Javascript クライアント [953](#)
- WebAuthn MDS [949](#)
- WebAuthn Registration Request [940](#)
- WebAuthn 登録 [937](#)
- WebAuthn 登録結果 [943](#)
- WebAuthn 認証 [944](#), [950](#)
- WebAuthn 認証 WebSocket [952](#)
- WebAuthn 認証リクエスト [946](#)
- WebAuthn 認証レスポンス [947](#)
- WebAuthn登録レスポンス [941](#)
- WebAuthn認証結果 [948](#)
- WebPush [681](#)
- WebRTC [1132](#)
- WebSocket [103](#)
- WebSocket イベント [103](#)
- WebSocket パラメーター接続 [104](#)
- WebSocket リダイレクション [156](#)
- WebSocket、チャンネル、Coinbase Pro サブスクライブ [457](#)
- WebSockets [66](#), [104](#), [148](#), [156](#), [170](#), [434](#), [435](#), [456](#), [491](#), [498](#)
- WebSockets Private [498](#)
- WebSockets Public [491](#)
- WebSockets、チャンネル、Forex サブスクライブ、Lightstreamer [588](#)
- WebView2 JavaScript [999](#)
- WebView2 Navigation [997](#)
- WebView2 クッキー [1001](#)
- WebView2 ダウンロード [1003](#)
- WebView2 設定 [1005](#)
- WebView2 高度な機能 [1007](#)
- WhatsApp Download Media [654](#)
- WhatsApp Send Interactive Messages [643](#)
- WhatsApp Webhook [638](#)
- WhatsApp アプリの作成 [634](#)
- WhatsApp セキュリティ [639](#)
- WhatsApp テンプレートメッセージの送信 [648](#)
- WhatsApp トークン [637](#)
- WhatsApp ファイル送信 [652](#)
- WhatsApp メッセージの受信とステータス通知 [650](#)
- WhatsApp メッセージの送信 [640](#)
- WhatsApp 電話番号 ID [636](#)
- Windows サービス [106](#)
- XSS [240](#)
- インストール [30](#)
- インラインボタン付き Telegram メッセージを送信する [664](#)
- エディション [28](#)
- エラー [207](#)
- カスタムオブジェクト [123](#)
- キュー [133](#)
- キュー、AMQP キュー [320](#)
- クライアント [149](#), [152](#), [161](#), [162](#), [170](#), [172](#), [196](#), [298](#), [300](#), [300](#), [314](#), [314](#), [334](#), [334](#), [411](#), [411](#), [849](#), [851](#), [852](#), [1026](#), [1116](#), [1116](#), [1117](#), [1120](#), [1125](#)
- ファイルの送信 [411](#)
- クライアント AMQP 接続 [314](#)
- クライアント AMQP1 Azure MessageBus [350](#)
- クライアント AMQP1 アイドルタイムアウト接続 [347](#)
- クライアント AMQP1 切断 [346](#)
- クライアント AMQP1 接続 [345](#)
- クライアント AMQP1 接続状態 [348](#)
- クライアント AMQP1 認証 [349](#)
- クライアント Kafka 切断 [334](#)
- クライアント MQTT セッション [300](#)
- クライアント MQTT 接続 [298](#)
- クライアント SocketIO [1120](#)
- クライアント WebSocket ハンドシェイク [170](#)
- クライアント バイナリメッセージの送信 [162](#)
- クライアント 接続を開いたまま保持 [152](#)
- クライアント 接続維持 [849](#)
- クライアント、STUN、STUNクライアント属性 [1018](#)

[クライアント、STUN、STUNクライアント長期認証情報](#) [1017](#)
[クライアント、Telegram クライアント](#) [1135](#)
[クライアント、クライアント AMQP 切断](#) [315](#)
[クライアント、クライアント、クライアント Kafka Connect、Kafka](#) [333](#)
[クライアントオープン接続](#) [149](#)
[クライアントスナップショット](#) [1125](#)
[クライアントチャット](#) [1116](#)
[クライアントのテキストメッセージ送信](#) [161](#)
[クライアントのテキスト送信](#) [161](#)
[クライアントプロキシ](#) [172](#)
[クライアントへのファイル送信](#) [411](#)
[クライアント一覧](#) [851](#)
[クライアント保留中のリクエスト](#) [851](#)
[クライアント接続](#) [190](#)
[クライアント接続の切断、クライアント](#) [151](#), [848](#)
[クライアント群](#) [849](#)
[クライアント認証](#) [852](#)
[クライアント資格情報グラント](#) [905](#)
[グループ](#) [124](#)
[クローラー検出](#) [246](#)
[コンシューマー](#) [336](#)
[コンシューマーグループ](#) [337](#)
[サーバー](#) [179](#), [188](#), [194](#), [196](#), [199](#), [201](#), [208](#), [844](#), [898](#), [1020](#), [1021](#), [1031](#), [1032](#), [1114](#)
[サーバー、サーバーSSL](#) [181](#)
[サーバー、サーバースナップショット](#) [1124](#)
[サーバー、サーバーテキストメッセージ送信](#) [192](#)
[サーバー、サーバーバイナリメッセージ送信](#) [193](#)
[サーバー、サーバーモニター](#) [1121](#)
[サーバー、サーバー接続のクローズ](#) [189](#)
[サーバーエンドポイント](#) [898](#)
[サーバーセッション](#) [208](#)
[サーバーチャット](#) [1114](#)
[サーバーの起動とシャットダウン](#) [179](#)
[サーバープレーン TCP](#) [188](#)
[サーバーヘッダーの読み取り](#) [196](#)
[サーバーリクエスト](#) [199](#)
[サーバー証明書の検証](#) [186](#)
[サービス](#) [204](#)
[品質](#) [131](#)
[サブスクライブ](#) [336](#)

[スロットル](#) [139](#)
[セキュアサーバーへの接続](#) [157](#)
[セキュア接続](#) [111](#)
[データセンター検出](#) [246](#)
[テキストメッセージの受信](#) [194](#)
[テキストメッセージを受信](#) [164](#)
[デスクトップアプリケーション向け OAuth2 クライアント](#) [873](#)
[デバイス認証グラント \(RFC 8628\)](#) [884](#), [911](#)
[ドロップされた切断](#) [153](#)
[バイナリメッセージの受信](#) [165](#)
[バインディング](#) [118](#)
[はじめに](#) [23](#)
[パッケージのインストール](#) [44](#)
[ビルド](#) [72](#)
[ビルド、OSX アプリケーションのビルド](#) [73](#)
[ファイル](#) [143](#), [411](#), [845](#)
[サーバー](#) [410](#)
[ファイル、ファイルのアップロード](#) [1126](#)
[ファイルの送信](#) [411](#)
[クライアント](#) [411](#)
[フラグメント化されたメッセージ](#) [145](#)
[フラッド保護](#) [243](#)
[ブルートフォース、自動禁止、IP 禁止、TsgcWebSocketFirewall](#) [238](#)
[フロー](#) [70](#)
[プロキシ](#) [144](#), [172](#)
[プロトコル](#) [371](#), [376](#), [384](#), [393](#), [406](#)
[プロトコル E2EE](#) [420](#)
[プロトコル WAMP](#) [376](#)
[プロトコル WAMP Javascript](#) [376](#)
[プロトコル WebRTC](#) [371](#)
[プロトコル WebRTC Javascript](#) [371](#)
[プロトコル、Protocol Presence、Protocol Presence Javascript](#) [417](#)
[プロトコル、デフォルトプロトコル](#) [388](#)
[プロトコル、プロトコル JavaScript](#) [292](#)
[プロトコル、プロトコル STOMP](#) [363](#)
[プロトコル、プロトコル WebRTC](#) [369](#)
[プロトコルデータセット、プロトコル](#) [397](#)
[プロトコルデータセット、プロトコル、プロトコルデータセット JavaScript](#) [401](#)
[プロトコルデータセット、プロトコル、プロトコルデータセットのテーブルレプリケーション](#) [404](#)
[プロトコルファイル](#) [406](#)

ポジション [585](#)

ボタン付き Telegram メッセージを送信する [664](#)

ボタン付きの Telegram メッセージを送信する [665](#)

ボット [667](#)

ボット検出 [246](#)

メモリマネージャー [82](#)

ライセンス [1140](#)

リソースオーナーパスワード認証情報グラント [907](#)

リバース DNS [246](#)

レート制限 [243](#)

レスポンスボディ [207](#)

ログ [115](#)

圧縮 [121](#)

埋め込み ChatBot [756](#)

埋め込みベクターの作成 [755](#)

履歴 [585](#)

概要 [64](#)

注文 [585](#)

消費 [336](#)

自己署名証明書 [221](#)

見つかりました [667](#)

証明書 SChannel [159](#)

認可コードグラント [903](#)

認証 [109](#)

高速パフォーマンスサーバー [78](#)

