SAS® Event Stream Processing: パブリッシュ/サブスクライブ API リファレンス

2023.01

このドキュメントは、ソフトウェアの追加バージョンに適用される場合があります。このドキュメントを SAS Help Center で開き、バナーのバージョンをクリックすると、使用できるすべてのバージョンが表示されます。

パブリッシュ/サブスクライブ API の概要

SAS Event Stream Processing は、Python、Java、C 向けのパブリッシュ/サブスクライブアプリケーションプログラミングインターフェース(API)を提供しています。これらの API を使って、次のことができます。

- 実行中のプロジェクトのソースウィンドウにイベントストリームをパブリッシュする
- プロジェクトのウィンドウイベントストリームをサブスクライブします

これらの API は、TCP/IP ネットワークをサポートし、IPv4 に対応しています。

これらの API を使用するアプリケーションを Kubernetes クラスタ内のポッドで実行すると、同じクラスタ内で動作するプロジェクトと通信することができます。

> **重要** これらの API は、Kubernetes クラスタの外部で動作するアプリケーションでは、クラスタ内で動作するプロジェクトへのパブリッシュやプロジェクトからのサブスクライブには使用できません。クラスタの外部で動作するアプリケーションは、代わりに WebSocket を使用して、クラスタ内で動作するプロジェクトにイベントを注入したり、イベントを照会したりすることができます。

ESP Connect API は、JavaScript オブジェクトを使用して、あらゆるプラットフォームから ESP サーバーと通信することができます。これらのオブジェクトは、Web ページに埋め込むことができ、また、Node.js をサポートしているため、コマンドラインから実行することもできます。この API は、

ESP のエッジサーバーだけでなく、Kubernetes 環境で動作する ESP サーバーとも連携します。その GitHub リポジトリから ESP Connect API にアクセスします。

Python パブリッシュ/サブスクライブ API の使用

SAS Event Stream Processing Python パブリッシュ/サブスクライブ API は、Python 3.4.*x* を使用して開発され、テストされました。他のバージョンでは、テストされておらずサポートされていません。

API は、Python の ctypes 外部関数ライブラリを使用して実装されています。このライブラリは Python で C パブリッシュ/サブスクライブ API をラップします。したがって、Python パブリッシュ/サブスクライブ API 定義は、C パブリッシュ/サブスクライブ API を反映します。関数は、C 関数と同じ名前を持ちますが、名前から C_dfESPpubsub が削除されています(C_dfESPpubsubInit が Init になるなど)。

API は、**pubsubApi.py** と **modelingApi.py** という 2 つのファイルで定義され、文書化されています。これらのファイルは、Linux システムの場合は**$DFESP_HOME/lib**、Windows システムの場合は**%DFESP_HOME%\lib** にあります。

オンライン.で Python で記述されたパブリッシャクライアントとサブスクライバクライアントのサンプルを見つけることができます。

Init()関数は、次の処理を行います。

- 必要な C API ライブラリを読み込みます
- Python の関数ポインタを設定します
- ESP ロガーインスタンスを作成します
- PythonInit()関数に渡されるロギングレベルとロギング設定ファイルパスを使用して C API C_dfESPpubsubInit() 関数を呼び出します
- C_dfESPpubsubInit()が返す戻り値を返します。

modelingApi.py には、C API 関数をラップしない次の余分な関数が含まれています。

- getLogger()
- arrayGetString(instance, i)
- arrayGetInt32(instance, i)

getLogger()は、Init()によって作成された Python ロガーインスタンスを返します。Python クライアントは次のようなロガーインスタンスを取得できます:

```
logger = logging.getLogger()

logger.addHandler(modelingApi.getLogger())
```

ロガーは標準の ESP ロギング機能への内部関数ポインタを定義しているため、ログメッセージは ESP サーバーまたは C クライアントによって記録されたメッセージのように見えます。logger.*level*(*string*) を呼び出すことによって、さまざまなレベルでメッセージを記録できます。*level* は **debug**、**info**、**warning**、**error**、または **critical** です。これらのレベルは、SAS Stream Processing ロギングレベル DEBUG、INFO、WARN、ERROR、および FATAL にそれぞれマップされます。

arrayGet*関数は、C 関数によって構築され、SchemaGetNames()などの Python 関数によって返された配列内のエントリを抽出するために使用されます。この抽出は、QueryMeta()によって返されたものな

ど、C 関数によって構築されたベクトルの処理とは異なります。このようなベクトルのエントリは、StringVGet()を呼び出すことで取得できます。

Java パブリッシュ/サブスクライブ API の使用

Java パブリッシュ/サブスクライブ API の概要

SAS Event Stream Processing と C パブリッシュ/サブスクライブ API は SAS ロギングライブラリを使用しますが、Java パブリッシュ/サブスクライブ API は **java.util.logging** パッケージの Java ロギング API を使用します。ログレベルと Java ロギングの詳細については、そのパッケージを参照してください。

Java パブリッシュ/サブスクライブ API は 2 つのパッケージで提供されています。これらのパッケージは、次のパブリックインターフェイスを定義します。

- **com.sas.esp.api.pubsub**
  - **com.sas.esp.api.pubsub.clientHandler**
  - **com.sas.esp.api.pubsub.clientCallbacks**
- **com.sas.esp.api.server**
  - **com.sas.esp.api.server.datavar**
  - **com.sas.esp.api.server.event**
  - **com.sas.esp.api.server.eventblock**
  - **com.sas.esp.api.server.library**
  - **com.sas.esp.api.server.schema**

クライアントは、イベントストリーム処理アプリケーションまたは ESP サーバーをいつでも照会して、現在実行中のウィンドウ、連続クエリ、およびプロジェクトをさまざまな粒度で検出できます。この情報は、名前を表す文字列のリストの形式でクライアントに返されます。このリストは、subscriberStart()または publisherStart()に渡す URL 文字列を作成するために使用できます。

注: 注:Java API に渡されるイベントストリームパブリッシュ URL が認証用に?*username* パラメータを含む場合、クラスパスに **cas-client-*.jar** を含める必要があります。

Java パブリッシュ/サブスクライブ API のパラメータと使用法は、C パブリッシュ/サブスクライブ API の同等の呼び出しと同じです。

C API リファレンスと Java インターフェースリファレンスは、**$DFESP_HOME/doc/html** にあります。

高レベルのパブリッシュ/サブスクライブメソッドの使用

次の高レベルパブリッシュ/サブスクライブメソッドは、**com.sas.esp.api.pubsub.clientHandler** で定義されています。

表 1 高レベルのパブリッシュ/サブスクライブメソッド

方法	説明
boolean init(Level *level*)	パブリッシュ/サブスクライブサービスを初期化します
dfESPclient publisherStart(String *serverURL*, clientCallbacks *userCallbacks*, Object *ctx*)	サイトパブリッシャを開始します。
dfESPclient subscriberStart(String *serverURL*, clientCallbacks *userCallbacks*, Object *ctx*)	サブスクライバを開始します
boolean connect(dfESPclient *client*)	イベントストリーム処理アプリケーションまたは ESP サーバーに接続します
boolean publisherInject((dfESPclient *client*, dfESPeventblock eventblock)	イベントブロックをパブリッシュします
ArrayList< String > *queryMeta* (String *queryURL*)	クエリモデルのメタデータ
ArrayList< String > getModel(String *queryURL*)	クエリモデルウィンドウとそのエッジ
boolean disconnect (dfESPclient *client*, boolean *block*)	イベントストリームプロセッサから切断します
boolean stop (dfESPclient *client*, boolean *block*)	サブスクライバまたはパブリッシャを停止します
void shutdown ()	パブリッシュ/サブスクライブサービスをシャットダウン
boolean setBufferSize(dfESPclient *client*, int *mbytes*)	デフォルトのソケット読み書きバッファサイズを変更する
dfESPclient GDsubscriberStart (String *serverURL*, clientCallbacks *userCallbacks*, Object *ctx*, String *configFile*)	保証された配信サブスクライバを開始します
dfESPclient GDpublisherStart (String *serverURL*, clientCallbacks *userCallbacks*, Object *ctx*, String *configFile*)	保証配信パブリッシャを開始します

方法	説明
long GDpublisherGetID()	順次配信されるイベントブロックに書き込むための ID を順次取得
boolean GDsubscriberAck(dfESPclient *client*, dfESPeventblock *eventblock*)	保証された配信通知をトリガします
boolean persistModel(String *hostportURL*, String *persistPath*)	実行中のエンジンに現在の状態をディスクに保存するよう指示します
boolean quiesceProject(String *projectURL*, dfESPclient *client*)	実行中のエンジンにモデル内の特定のプロジェクトを休止するよう指示します。
boolean subscriberMaxQueueSize(String *serverURL*, int *maxSize*, boolean *block*)	サブスクライバに送信されたイベントブロックをエンキューするために使用される、イベントストリーム処理サーバー内のキューの最大サイズを構成します。キューのサイズが **maxSize** を下回るのを待つために **block** を 1 に設定し、そうでなければクライアントを切断します。
boolean pingHostPort(String *hostportURL*)	実行中のエンジンに ping を実行して、指定されたパブリッシュ/サブスクライブポートが開いているかどうかを確認します。また、オープンポートがパブリッシュ/サブスクライブポートであることを確認するために、マジックナンバーを交換して確認します。
boolean setTokenLocation(String *tokenLocation*)	パブリッシュ/サブスクライブサーバーによる認証に必要な OAuth トークンを含むローカルファイルシステム内のファイルの場所を設定します。
void setTransportConfigFile(String *xportCfgFile*)	ローカルファイルシステム内のパブリッシュ/サブスクライブトランスポート構成ファイルの場所を設定します。使用されている代替のトランスポートライブラリに応じて、デフォルト値は **"kafka.cfg"** 、 **"rabbitmq.cfg"** 、 **"solace.cfg"** 、または**"client.config"**です。

詳細は、 **$DFESP_HOME/doc/html/index.html** を参照してください。**クライアントハンドラ**については、**クラス**ページを検索してください。

Google プロトコルバッファをサポートするメソッドの使用

次のメソッドが **com.sas.esp.api.pubsub.protobufInterface** に定義されています。

表 2 Google プロトコルバッファをサポートするメソッド

方法	説明
boolean init(String *fileDescriptorSet*, String *msgName*, dfESPschema *schema*, EventOpcodes *defaultOpcode*)	Google プロトコルバッファをサポートするライブラリを初期化します。
dfESPeventblock protobufToEb(byte[] *serializedProtobuf*)	protobuf メッセージをイベントブロックに変換します。
byte[] ebToProtobuf(dfESPeventblock *eb*, int *index*)	イベントブロック内のイベントを protobuf メッセージに変換します。

詳細については、“Google プロトコルバッファのパブリッシュ/サブスクライブ API サポート”を参照してください。

ユーザー提供のコールバック関数の使用

com.sas.esp.api.pubsub.clientCallbacks インタフェースリファレンスは、ユーザー提供のコールバック関数のシグネチャを定義します。現在、次の 3 つの機能があります。

- サブスクライブされたイベントブロックハンドラ
- パブリッシュ/サブスクライブ失敗ハンドラ
- 保証配信 ACK-NACK ハンドラ

サブスクライブされたイベントブロックハンドラは、サブスクライバクライアントによってのみ使用されます。これは、アプリケーションまたはサーバーからの新しいイベントブロックが到着したときに呼び出されます。イベントブロックを処理した後、クライアントは eventblock_destroy()を呼び出して解放します。

このユーザー定義のコールバックのシグネチャは次のとおりです。

void com.sas.esp.api.pubsub.clientCallbacks.dfESPsubscriberCB_func
(dfESPeventblock *eventBlock*, dfESPschema *schema*, Object *ctx*)

- *eventBlock* は読み込まれたイベントブロックです
- *schema* は、クライアント処理用のイベントのスキーマです。
- *ctx* は、コール状態を維持するためのオプションのコンテキストポインタです。

パブリッシュ/サブスクライブクライアント ERROR 処理の 2 番目のコールバック関数は、サブスクライバクライアントとパブリッシャクライアントの両方でオプションです。指定されている場合(つま

りヌルでない場合)、クライアントサービス内の異常な切断のような異常なイベントが発生するたびに呼び出されます。これにより、クライアントは正常に処理し、場合によってはパブリッシュ/サブスクライブサービス ERROR から回復することができます。このコールバック関数のシグネチャは次のとおりです。

- failure は pubsubFail_APIFAIL、pubsubFail_THREADFAIL、または pubsubFail_SERVERDISCONNECT です。
- code は、失敗の特定のコードを提供します。
- ctx は、状態データ構造体へのオプションのコンテキストポインタです。

```
void com.sas.esp.api.pubsub.clientCallbacks.dfESPpubsubErrorCB_func
    (clientFailures failure, clientFailureCodes code, Object ctx)
```

clientFailures および client FailureCodes は、インタフェース参照 **com.sas.esp.api.pubsub.clientFailures** および **com.sas.esp.api.pubsub.clientFailureCodes** で定義されています。

保証された配信 ACK-NACK ハンドラは、特定のイベントブロックのステータスを提供するために、またはすべてのサブスクライバが接続され、パブリッシュが開始できることをパブリッシャに通知するために呼び出されます。このコールバック関数のシグネチャは次のとおりです。

```
void com.sas.esp.api.pubsub.clientCallbacks.dfESPGDpublisherCB_func
  (clientGDStatus eventBlockStatus, long eventBlockID, Object ctx)
```

どこで

- eventBlockStatus は、ESP_GD_READY、ESP_GD_ACK、または ESP_GD_NACK です。
- eventBlockID は、パブリッシュ前にイベントブロックに書き込まれた ID です。
- ctx は、状態データ構造体へのオプションのコンテキストポインタです。

Java クライアント用の代替トランスポートの使用

通常、Java パブリッシュ/サブスクライブクライアントは、ESP サーバーとの通信に直接 TCP/IP 接続を使用します。次のライブラリは、Java パブリッシュ/サブスクライブクライアントが別のメカニズムでイベントブロックを送受信することを可能にします。

- RabbitMQ Java ライブラリは、RabbitMQ サーバーを介して送受信できます。
- Solace Java ライブラリを使用すると、Solace アプライアンスを介して送受信できます。
- Kafka Java ライブラリは、Kafka クラスタを介して送受信できます。

これらのライブラリは、Java の C パブリッシュ/サブスクライブ API メソッドと同等のものを使用して、代替トランスポートを置き換えます。エンジンが RabbitMQ、Solace、または Kafka を使用して 1+N-Way フェイルオーバー用に構成されている場合、Java パブリッシュ/サブスクライブクライアントは対応するクライアントライブラリを使用してフェイルオーバーを success させる必要があります。

Java パブリッシュ/サブスクライブクライアントアプリケーションでこれらのライブラリの 1 つを置き換えるには、次のように、クラスパスの **dfx-esp-api.jar** の前に対応する JAR filename を挿入します。

- RabbitMQ の場合、JAR ファイルは **dfx-esp-rabbitmq-api.jar** です。
- Solace の場合、JAR ファイルは **dfx-esp-solace-api.jar** です。
- Kafka の場合、JAR ファイルは **dfx-esp-kafka-api.jar** です。

RabbitMQ ライブラリを使用する場合は、ネイティブ RabbitMQ Java クライアントライブラリ(**amqp-client-3.4.2.jar**)もインストールする必要があります。http://www.rabbitmq.com/java-client.html で

入手してください。次に、 **$DFESP_HOME/lib** から **commons-configuration-1.10.jar** 、**commons-lang-2.6.jar** 、および **commons-logging-1.2.jar** と一緒に **amqp-client-3.4.2.jar** をクラスパスに追加します。

1+N-Way フェイルオーバーをサポートする Java パブリッシャアダプタに Kafka ライブラリを使用する場合は、Apache Zookeeper Java クライアントライブラリもインストールする必要があります。http://zookeeper.apache.org/releases.html で入手し、クラスパスに追加します。

現在の作業ディレクトリには、パブリッシュ/サブスクライブクライアントと ESP サーバー間の通信を管理するための構成ファイルも含まれている必要があります。

- RabbitMQ の場合、このファイルの名前は **rabbitmq.cfg** でなければなりません。
- Solace の場合は、 **solace.cfg** という名前にする必要があります。
- Kafka の場合、**kafka.cfg** という名前にする必要があります。

または、setTransportConfigFile()メソッドを使用して、別のディレクトリにある構成ファイルの絶対パスを指定することもできます。

RabbitMQ の設定ファイルの例を次に示します。

```
{
rabbitmq =
{
host = "my.machine.com";
port = "5672";
exchange = "SAS";
userid = "guest";
password = "encrypted_password_generated_by_OpenSSL";
passwordencrypted = true;
heartbeat = "5"
}
sas =
{
buspersistence = true;
queuename = "subpersist";
protobuf = true;
descfile = "./GpbHistSimFactory.desc";
protomsg = "GpbTrade";
noreplay = true;
}
}
```

次のことに注意してください。

- host、port、exchange、userid、および password パラメータが必要です。
- passwordencrypted パラメータはオプションで、デフォルト値は **false** です。

  注: passwordencrypted = true の場合、**password** の値は、この OpenSSL コマンドで生成された暗号化されたパスワードでなければなりません。

  ```
  echo "password" | openssl enc -e -aes-256-cbc -a -salt
  -pass pass:"espRMQclientUsedByUser=userid"
  ```

- heartbeat パラメータはオプションで、デフォルトでは heartbeat はありません。
- buspersistence パラメータはオプションで、デフォルト値は **false** です。
- queuename パラメータは、サブスクライバの場合は必要で、パブリッシャの場合は無視されます。
- protobuf パラメータはオプションで、デフォルト値は **false** です。

- descfile パラメータは、protobuf=true の場合に必要です。
- protomsg パラメータが必要です。
- noreplay パラメータはオプションで、デフォルト値は **false** です。**true** に設定すると、buspersistence が有効になっていても、受信したメッセージが肯定応答されます。

buspersistence パラメータと queuename パラメータは、パブリッシャとサブスクライバにとって異なる意味を持ちます。

- パブリッシャの場合、queuename は常に無視されます。buspersistence = false の場合、メッセージは非永続配信モードで送信されます。それ以外の場合、配信モードは永続的です。
- サブスクライバにとって、buspersistence = false は、クライアントによって作成されたすべてのキューおよびエクスチェンジが耐久性がなく、自動削除され、queuename パラメータが無視されることを意味します。buspersistence = true の場合、すべての交換とキューは永続的で自動削除ではなく、耐久性のある受信キューの queuename は固定されます。

Solace の設定ファイルの例を次に示します。

```
{
solace =
{
session = ( "host", "10.37.150.244:55555",
"username", "sub1", "encrypted_password",
"sub1", "vpn_name", "SAS"); (all required)
context = ( "CONTEXT_TIME_RES_MS", "50",
"CONTEXT_CREATE_THREAD", "1" );
}
sas=
{
buspersistence = true;
queuename = "myqueue";
protobuf = true;
descfile = "./GpbHistSimFactory.desc";
protomsg = "GpbTrade";
passwordencrypted = true;
}
}
```

次のことに注意してください。

- session および context パラメータが必要です。
- buspersistence パラメータはオプションで、デフォルト値は **false** です。
- queuename パラメータは、サブスクライバの場合で buspersistence=true の時は必要で、パブリッシャの場合は無視されます。
- protobuf パラメータはオプションで、デフォルト値は **false** です。
- descfile パラメータは、protobuf=true の場合に必要です。
- protomsg パラメータは、protobuf=true の場合に必要です。
- passwordencrypted はオプションです。デフォルト値は **false** です。

注: passwordencrypted = true の場合、**session.password** の値は、この OpenSSL コマンドで生成された暗号化されたパスワードでなければなりません。

```
echo "session.password" | openssl enc -e -aes-256-cbc -a -salt
-pass pass:"espSOLclientUsedByUser=session.username"
```

注: PASSWORDENCRYPTED = 1 の場合、**PASSWORD** の値はこの OpenSSL コマンドで生成された暗号化されたパスワードでなければなりません。

```
echo "PASSWORD" | openssl enc -e -aes-256-cbc -a -salt
-pass pass:"espTVAclientUsedByUser=USERNAME"
```

Kafka の設定ファイルの例を次に示します。

```
{
kafka =
{
hostport = "kafkahost:9092"
partition = "0"
initialoffset = "largest"
groupid = "mygroup"
metatopic = "mymetatopic"
session_timeout_ms = "40000")
}
sas =
{
protobuf = true;
descfile = "./GpbHistSimFactory.desc";
protomsg = "GpbTrade";
hotfailover = false;
numbufferedmsgs = "16000"
zookeeperhostport = "myhost:myport"
failovergroup = "mygroup"
}
}
```

次のことに注意してください。

- hostport、partition、および initialoffset パラメータが必要です。
- groupid パラメータは、default=null のサブスクライバのオプションです。パブリッシャは無視されます。
- metatopic パラメータはオプションです。デフォルト値は *random_string* です。
- session_timeout_ms パラメータはオプションです。デフォルト値は **30000** です。
- protobuf パラメータはオプションです。デフォルト値は **false** です。
- descfile パラメータは、protobuf=true の場合に必要です。
- protomsg パラメータは、protobuf=true の場合に必要です。
- hotfailover パラメータは、デフォルト値が **false** のパブリッシャにはオプションです。サブスクライバは無視されます。
- numbufferedmsgs パラメータは、hotfailover=true の場合に必要です。
- zookeeperhostport パラメータは、hotfailover=true の場合に必要です。
- failovergroup は、hotfailover=true の場合に必要です。

C パブリッシュ/サブスクライブ API の使用

エンジンの視点からの C パブリッシュ/サブスクライブ API

C++モデリング API を使用してエンジンインスタンスのパブリッシュ/サブスクライブを有効にするには、次のように、dfESPengine::initialize()コールの pubsub_ENABLE()パラメータにポート番号を指定する必要があります。

```
dfESPengine *engine;
engine = dfESPengine::initialize(argc, argv, "engine",
pubsub_ENABLE(33335));

if (engine == NULL) {
    cerr <<"Error: dfESPengine::initialize() failed\n";
    return 1;
}
```

注: ESP サーバーでは、コマンドラインで"–pubsub *port* "を指定するか、エンジンエレメントのポート属性を使用してポートを指定します。

注: 単一のパブリッシュ/サブスクライブポートしか使用できない場合は、single_port_mode を **esp-properties.yml** で true に設定します。

クライアントは、ポート番号(この例では 33335)を使用してパブリッシュ/サブスクライブ接続を確立できます。パブリッシュ/サブスクライブが不要な場合は、そのパラメータに pubsub_DISABLE を使用します。

パブリッシュ/サブスクライブが必要で、クライアントを認証する必要がある場合は、次のいずれかを使用します。

- pubsub_ENABLE_OAUTH(port, *clientId*) (ここで、 *clientId* は CF UAA OAuth サーバークライアント ID です)
- pubsub_ENABLE_SASLOGONOAUTH(port, *sasLogonURL*) (ここで、 *sasLogonURL* は SASLogon サービスの基本サービス URL です)
- pubsub_ENABLE_KERBEROS(port, *serviceName*) (ここで、*serviceName* は Kerberos サービスプリンシパル名のサービス名部分です。)

ESP サーバーの場合は、**-auth oauth://*clientId*** または**-auth saslogon://*sasLogonURL*** または**-auth kerberos://*serviceName*** を指定します。

詳細については、"TCP/IP 接続での認証の有効化" (*SAS Event Stream Processing: セキュリティの実装*)を参照してください。

プロジェクトのパブリッシュ/サブスクライブ機能を初期化するには、engine->startProjects() を呼び出す前に project->setPubSub() を呼び出します。

たとえば、

```
project->setPubSub(dfESPproject::ps_AUTO);
engine->startProjects();
```

注: ESP サーバーと同等の機能は、**プロジェクト**エレメントの **pubsub** 属性を使用してパブリッシュ/サブスクライブ機能を指定することです。

このコードは、ポート 33335 でサーバーリスナーソケットを開き、クライアントのサブスクライバとパブリッシャがパブリッシュ/サブスクライブサービス用に ESP サーバーに接続できるようにします。クライアントからのパブリッシュ/サブスクライブの接続要求が(後述のように)行われた後、パブリッシュ/サブスクライブ API がこの接続に使用する一時的なポートが返されます。

特定のポートの一時ポートを(セキュリティ上の目的で)オーバーライドするには、このプロジェクトへの接続に使用する優先ポートである第 2 パラメータを使用して project->setPubSub を指定します。

たとえば、

```
project->setPubSub(dfESPproject::ps_AUTO, 33444);
```

注: ESP サーバーと同等の機能は、**プロジェクト**エレメントの**ポート**属性を使用してポートを指定することです。

project->setPubSub()の最初のパラメータはサブスクリプションサービスにのみ適用され、プロジェクトのウィンドウがクライアントのサブスクリプションをサポートする方法を指定します。ps_AUTO を指定すると、クライアントはプロジェクト内のすべてのウィンドウ出力イベントストリームをサブスクライブすることができます。

または、ps_MANUAL を指定して手動でウィンドウを有効にすることもできます。重要ではないプロジェクトでは、すべてのウィンドウを 自動的に有効にすることが全体のパフォーマンスに顕著な影響を与えるため、手動で任意のウィンドウを有効にします。ps_NONE を指定することもできます。これにより、すべてのウィンドウのサブスクライブを 無効にします。

project->setPubSub() で ps_MANUAL を使用してウィンドウサブスクリプションを手動で有効にする場合は、それぞれのウィンドウに対して enableWindowSubs() を使用して、サブスクライブを次のように有効にします。

```
project->enableWindowSubs(dfESPwindow *w);
```

注: ESP サーバーは、**ウィンドウ**エレメントの **pubsub** 属性を使用して、パブリッシュ/サブスクライブを有効または無効にします。

ただし、setPubSub()で ps_AUTO または ps_NONE を指定した場合、その後の enableWindowSubs() の呼び出しは無視され、警告が生成されます。

注: クライアントは、現在実行中のプロジェクトの任意のソースウィンドウ(およびソースウィンドウのみ)にイベントストリームをパブリッシュできます。デフォルトでは、すべてのソースウィンドウがパブリッシュ用に有効になっています。

クライアントの視点からの C パブリッシュ/サブスクライブ API

C パブリッシュ/サブスクライブ API を使用してエンジンのイベントストリームをサブスクライブしたりパブリッシュしたりするクライアントは、次を行う必要があります。

1 C_dfESPpubsubInit()を使用してクライアントのサービスを初期化します。

2 C_dfESPsubscriberStart()を使用してサブスクリプションを開始し、C_dfESPpublisherStart()を使用してパブリッシャを開始します。

3 C_dfESPpubsubConnect()を使用してアプリケーションまたはサーバーに接続します。

パブリッシャを実装するクライアントは、C_dfESPpublisherStart().に渡された URL で指定されたソースウィンドウにイベントブロックをパブリッシュするために、必要に応じて C_dfESPpublisherInject() を呼び出すことができます。

クライアントパブリッシュ/サブスクライブ API の詳細は次のとおりです。

- パブリッシャサービスとサブスクライバサービスを提供するには、クライアントアプリケーションにヘッダーファイル C_dfESPpubsubApi.h を含める必要があります。API 呼び出しに加えて、このファイルにはユーザー提供のコールバック関数のシグニチャも定義されています。その中には、サブスクライブされたイベントブロックハンドラーとパブリッシュ/サブスクライブエラーハンドラーの 2 つがあります。
- サブスクライブされたイベントブロックハンドラは、サブスクライバクライアントによってのみ使用されます。これは、アプリケーションまたはサーバーからの新しいイベントブロックが到着したときに呼び出されます。イベントブロックを処理した後、クライアントは C_dfESPeventblock_destroy()を呼び出して解放します。このユーザー定義のコールバックのシグネチャは、次のとおりです。"eb"は読み込んだイベントブロック、"schema"はクライアント処理のイベントのスキーマ、ctx はコール状態を含むオプションのコンテキストオブジェクトです。

  ```
  typedef void (*C_dfESPsubscriberCB_func)(C_dfESPeventblock eb,
    C_dfESPschema schema, void *ctx);
  ```

- 2 番目のコールバック関数 C_dfESPpubsubErrorCB_func()は、サブスクライバクライアントとパブリッシャクライアントの両方でオプションです。指定されている場合(つまりヌルでない場合)、クライアントサービス内の異常な切断のような異常なイベントが発生するたびに呼び出されます。これにより、クライアントはパブリッシュ/サブスクライブ・サービスの ERROR を処理し、場合によってはリカバリできます。このコールバック関数のシグネチャは次のとおりです。
  - failure は pubsubFail_APIFAIL、pubsubFail_THREADFAIL 、または pubsubFail_SERVERDISCONNECT です。
  - code は障害の特定のコードを提供します
  - ctx は、コール状態を含むオプションのコンテキストオブジェクトです

    ```
    typedef void (*C_dfESPpubsubErrorCB_func)(C_dfESPpubsubFailures
      failure, C_dfESPpubsubFailureCodes code);
    ```

- C_dfESPpubsubFailures および C_dfESPpubsubFailureCodes 列挙体は、C_dfESPpubsubFailures.h で定義されています。
- パブリッシャクライアントは、C_dfESPpublisherInject()API 関数を使用して、イベントブロックをアプリケーションまたはサーバーのソースウィンドウにパブリッシュします。イベントブロックは、C_dfESPpublisherStart()に渡された URL で指定された連続クエリおよびプロジェクトで実行されているソースウィンドウに挿入されます。クライアントは、各ウィンドウに対して

C_dfESPpublisherStart()を 1 回呼び出してから、適切なクライアントオブジェクトを必要に応じて C_dfESPpublisherInject()に渡すことで、プロジェクト内の複数のウィンドウにイベントをパブリッシュできます。

- クライアントは、いつでも現在実行中のウィンドウ、連続したクエリ、およびプロジェクトをさまざまなレベルで検出するために、アプリケーションまたはサーバーにいつでも問い合わせることができます。この情報は、名前を表す文字列のリストの形式でクライアントに返されます。これらの文字列は、C_dfESPsubscriberStart()または C_dfESPpublisherStart()に渡す URL を作成するために引き続き使用される場合があります。サポートされる照会のリストについては、関数の説明を参照してください。

C パブリッシュ/サブスクライブ API の一般的な関数

C_dfESPpubsubInit

この関数は、クライアントパブリッシャサービスとサブスクライバサービスを初期化します。C_dfESPpubsubSetPubsubLib()および C_dfESPpubsubSetTransportConfigFile()を除き、他のクライアント呼び出しを行う前に(一度だけ)呼び出すことができます。

int **C_dfESPpubsubInit**(C_dfESPLoggingLevel *level*, const char **logConfigPath*)

表 3 C_dfESPpubsubInit のパラメータ

パラメータ	説明
level	ログレベル
logConfigPath	ログ設定ファイルのフルパス名

表 4 C_dfESPpubsubInit の戻り値

値	説明
1	成功
0	失敗 — エラーはログに書き出されます。

C_dfESPpublisherStart

この関数は、特定のパブリッシャクライアント接続の接続パラメータを検証して保持します。

clientObjPtr **C_dfESPpublisherStart** (char**serverURL*, C_dfESPpubsubErrorCB_func *errorCallbackFunction* , void **ctx*)

表 5 C_dfESPpublisherStart のパラメータ

パラメータ	説明
serverURL	宛先ホスト、ポート、プロジェクト、連続問合せ、およびウィンドウを表す文字列。次の形式を使用します。 "dfESP:// *host*:*port* /*project*/ *contquery*/*window* "
errorCallbackFunction	クライアントサービスの障害を処理するためのヌルまたはユーザー定義の関数ポインタ
ctx	この呼び出しに状態を渡すためのオプションのコンテキストポインタ

戻り値は、次に説明するすべての API 関数に渡されるクライアントオブジェクトへのポインタ、または失敗した場合はヌルです。

C_dfESPGDpublisherStart

clientObjPtr **C_dfESPGDpublisherStart** ()

この関数は、パラメータと戻り値が C_dfESPpublisherStart() と同じです。

さらに 2 つの必須パラメータがあります。型が C_dfESPGDpublisherCB_func の保証配信コールバック関数ポインタとこのパブリッシャの保証配信設定ファイルのファイル名。

C_dfESPsubscriberStart

この関数は、特定のサブスクライバクライアント接続の接続パラメータを検証して保持します。

clientObjPtr **C_dfESPsubscriberStart**(char **serverURL*, C_dfESPsubscriberCB_func *callbackFunction* , C_dfESPpubsubErrorCB_ func *errorCallbackFunction* <, void* *ctx*>)

表 6 C_dfESPsubscriberStart のパラメータ

パラメータ	説明
serverURL	宛先ホスト、ポート、プロジェクト、連続問合せ、およびウィンドウを表す文字列。また、クライアントのスナップショット要件を指定します。クライアントが増分更新の前にウィンドウの現在のスナップショット状態を受け取ると、"true"になります。 クライアントの折り畳み要件の指定はオプションです。デフォルトでは false です。true の場合、UPDATE_BLOCK イベントは UPDATE イベントに変換され、サブスクライバの出力をパブリッシュ可能にします。 次の形式を使用します。 "dfESP:// *host*:*port* /*project*/ *contquery*/*window* ?snapshot=true |false <?collapse=true | false ><?rmretdel=true | false>"
callbackFunction	受信したイベントブロックを処理するためのユーザー定義関数へのポインタ。この関数は、C_dfESPeventblock_destroy()を呼び出してイベントブロックを解放する必要があります。

パラメータ	説明
errorCallbackFunction	サブスクリプションサービスの失敗を処理するためのヌルまたはユーザー定義の関数ポインタ
ctx	この呼び出しに状態を解析するためのオプションのコンテキストポインタ

戻り値は、次に説明するすべての API 関数に渡されるクライアントオブジェクトへのポインタ、または失敗した場合はヌルです。

C_dfESPGDsubscriberStart

clientObjPtr **C_dfESPGDsubscriberStart** ()

この関数は、パラメータと戻り値が C_dfESPsubscriberStart() と同じです。1 つの追加必須パラメータがあります。このサブスクライバの保証配信設定ファイルのファイル名。

C_dfESPpubsubConnect

この関数を使用して、アプリケーションまたはサーバーとの接続を確立します。

int **C_dfESPpubsubConnect**(clientObjPtr *client*)

表 7 C_dfESPpubsubConnect のパラメータ

パラメータ	説明
client	C_dfESPsubscriberStart() 、C_dfESPpublisherStart()、“ C_dfESPGDsubscriberStart”、“ C_dfESPGDpublisherStart”のいずれかによって返されたクライアントオブジェクトへのポインタ

表 8 C_dfESPpubsubConnect の戻り値

値	説明
1	成功
0	失敗 — エラーはログに書き出されます。

C_dfESPpubsubDisconnect

この関数は、渡されたクライアントオブジェクトに関連付けられている接続を閉じます。

int **C_dfESPpubsubDisconnect**(clientObjPtr *client* , int *block*)

表 9 C_dfESPpubsubDisconnect のパラメータ

パラメータ	説明
client	C_dfESPsubscriberStart() 、C_dfESPpublisherStart()、“ C_dfESPGDsubscriberStart”、“ C_dfESPGDpublisherStart”のいずれかによって返されたクライアントオブジェクトへのポインタ
block	キューに入れられたすべてのイベントが処理されるのを待つ場合は 1 に設定し、それ以外の場合は 0 に設定します

表 10 C_dfESPpubsubDisconnect の戻り値

値	説明
1	成功
0	失敗 — エラーはログに書き出されます。

C_dfESPpubsubStop

この関数は、クライアントセッションを停止し、渡されたクライアントオブジェクトを削除します。

int **C_dfESPpubsubStop**(clientObjPtr *client* , int *block*)

表 11 C_dfESPpubsubStop のパラメータ

パラメータ	説明
client	C_dfESPsubscriberStart() 、C_dfESPpublisherStart()、“ C_dfESPGDsubscriberStart”、“ C_dfESPGDpublisherStart”のいずれかによって返されたクライアントオブジェクトへのポインタ
block	キューに入れられたすべてのイベントが処理されるのを待つ場合は 1 に設定し、それ以外の場合は 0 に設定します

表 12 C_dfESPpubsubStop の戻り値

値	説明
1	成功
0	失敗 — エラーはログに書き出されます。

C_dfESPpublisherInject

イベントをパブリッシュする関数です。イベントブロックは、イベントストリームプロセッサのオブジェクト C API で提供される他の追加機能を使用して構築できます。

int **C_dfESPpublisherInject**(clientObjPtr *client* , C_dfESPeventblock *eventBlock*)

表 13 C_dfESPpublisherInject のパラメータ

パラメータ	説明
client	C_dfESPpublisherStart()または“ C_dfESPGDsubscriberStart”によって返されたクライアントオブジェクトへのポインタ
eventBlock	注入するイベントブロック。ブロックは、渡されたクライアントオブジェクトに関連付けられたソースウィンドウ、連続クエリ、およびプロジェクトに挿入されます。

表 14 C_dfESPpublisherInject の戻り値

値	説明
1	成功
0	失敗 — エラーはログに書き出されます。

C_dfESPpubsubQueryMeta

この関数は、独立したソケットを開いてクエリを送信し、クエリ応答を受信するとソケットを閉じます。

この関数は、一般的なイベントストリームプロセッサのメタデータ問い合わせメカニズムを実装します。このメカニズムにより、クライアントは、現在実行されているプロジェクト、連続クエリ、ウィンドウ、ウィンドウスキーマ、およびウィンドウのエッジを検出できます。このメカニズムは、クライアントによって 実行される他のアクティビティとの 依存関係も相互作用もありません。

C_dfESPstringV **C_dfESPpubsubQueryMeta** (char **queryURL*)

queryURL の ESP サーバーに送信されるクエリを表す文字列を指定します。

表 15 C_dfESPpubsubQueryMeta でサポートされている *queryURL* の形式

形式	結果
"dfESP:// *host*:*port* ?get=projects"	現在実行中のプロジェクトの名前を返します。
“dfESP:// *host*:*port* ?get=projects_pubsubonly”	パブリッシュ/サブスクライブが有効になっている

形式	結果
	現在実行中のプロジェクトの名前を返します。
"dfESP:// *host*:*port* ?get=queries"	現在実行中のプロジェクトで連続クエリの名前を返します。
"dfESP:// *host*:*port* /*project*?get=windows_sourceonly"	プロジェクトが実行中の場合、指定されたプロジェクトのソースウィンドウの名前を返します
"dfESP:// *host*:*port* /*project*?get=windows_derivedonly"	プロジェクトが実行中の場合、指定されたプロジェクトの派生ウィンドウの名前を返します。
"dfESP:// *host*:*port* ?get=queries_pubsubonly"	現在実行中のプロジェクトでパブリッシュ/サブスクライブが可能なウィンドウを含む連続クエリーの名前を返します
"dfESP:// *host*:*port* ?get=windows"	現在実行中のプロジェクトのウィンドウの名前を返します。
"dfESP:// *host*:*port* ?get=windows_pubsubonly"	現在実行中のプロジェクトのパブリッシュ/サブスクライブ可能なウィンドウの名前を返します。
"dfESP:// *host*:*port* ?get=windowsandtypes"	現在実行中のプロジェクトのウィンドウの名前を返します。返される各ウィンドウ名の後にコロンとウィンドウタイプが続きます
"dfESP:// *host*:*port* /*project*/ *contquery*?get=windows_sourceonly"	プロジェクトが実行中の場合、指定された連続したクエリおよびプロジェクト内のソースウィンドウの名前を返します
"dfESP:// *host*:*port* /*project*/ *contquery*?get=windows_derivedonly"	プロジェクトが実行されている場合、指定された連続したクエリおよびプロジェクト内の派生ウィンドウの名前を返します

形式	結果
"dfESP:// *host*:*port* /*project*?get=windows"	実行中の場合、指定されたプロジェクト内のウィンドウの名前を返します。
"dfESP:// *host*:*port* /*project*?get=windowsandtypes"	実行中の場合、指定されたプロジェクト内のウィンドウの名前を返します。返される各ウィンドウ名の後にコロンとウィンドウタイプが続きます
"dfESP:// *host*:*port* /*project*?get=windows_pubsubonly"	実行中の場合、指定されたプロジェクトのパブリッシュ/サブスクライブ可能なウィンドウの名前を返します。
"dfESP:// *host*:*port* /*project*?get=queries"	実行中の場合、指定されたプロジェクト内の連続するクエリの名前を返します
"dfESP:// *host*:*port* /*project*?get=queries_pubsubonly"	実行中の場合、指定されたプロジェクトにパブリッシュ/サブスクライブ可能なウィンドウを含む連続クエリの名前を返します
"dfESP:// *host*:*port* /*project*/ *contquery*?get=windows"	実行中の場合、指定された連続したクエリおよびプロジェクト内のウィンドウの名前を返します
"dfESP:// *host*:*port* /*project*/ *contquery*?get=windowsandtypes"	実行中の場合は、指定された連続クエリおよびプロジェクト内のウィンドウの名前を返します。返される各ウィンドウ名の後にコロンとウィンドウタイプが続きます。
"dfESP:// *host*:*port* /*project*/ *contquery*?get=windows_pubsubonly"	実行中の場合、指定された連続したクエリおよびプロジェクト内のパブリッシュ/サブスクライブ可能なウィンドウの名前を返します
"dfESP:// *host*:*port* /*project*/ *contquery*/*window* ?get=schema"	ウィンドウスキーマのシリアル化されたバージョ

形式	結果
	ンである単一の文字列を返します。
"dfESP:// *host:port /project/ contquery/window* ?get=edges"	すべてのウィンドウのエッジの名前を返します
"dfESP:// *host:port /project/ contquery/window* ?get=rowcount"	現在ウィンドウ内にある行の数を返します。

戻り値は、クエリへのレスポンスを構成する名前のリストを表す文字列のベクトル、または失敗した場合はヌルです。リストの終わりは空の文字列で示されます。呼び出し元は、C_dfESPstringV_free() を呼び出してベクトルを解放します。

C_dfESPpubsubGetModel

この関数を使用すると、モデルまたはプロジェクトまたは連続クエリのウィンドウの完全なセットをウィンドウのエッジと共に返します。クライアントによって 実行される他のアクティビティとの 依存関係や相互作用はありません。クエリを送信するために独立したソケットを開き、クエリ応答を受信するとソケットを閉じます。

C_dfESPstringV **C_dfESPpubsubGetModel** (char **queryURL*)

queryURL のエンジンに送信されるクエリを表す文字列を指定します。

表 16 C_dfESPpubsubGetModel でサポートされている *queryURL* の形式

形式	結果
"dfESP:// *host:port* "	モデル内のすべてのウィンドウの名前とそのエッジを返します
"dfESP:// *host:port /project*"	プロジェクト内のすべてのウィンドウの名前とそのエッジを返します
"dfESP:// *host:port /project/ contquery*"	連続クエリ内のすべてのウィンドウの名前とそのエッジを返します

戻り値は、クエリへのレスポンスを表す文字列のベクトル、または失敗した場合はヌルです。各文字列の形式は *"project/query/window:edge1, edge2 , ..."*です。リストの終わりは空の文字列で示されます。呼び出し元は、C_dfESPstringV_free()を呼び出してベクトルを解放します。

C_dfESPpubsubShutdown

void **C_dfESPpubsubShutdown**()

パブリッシュ/サブスクライブサービスを停止します。

C_dfESPpubsubPersistModel

この関数は、*hostportURL* のエンジンに現在の状態をディスクに保持するよう指示します。この関数には、クライアントによって実行される他のアクティビティとの 依存関係や相互作用はありません。要求を送信するために独立したソケットを開き、要求戻りコードを受け取るとソケットを閉じます。

int **C_dfESPpubsubPersistModel**(char **hostportURL*, const char * *persistPath*

表 17 C_dfESPpubsubPersistModel のパラメータ

パラメータ	説明
hostportURL	"dfESP://*host*: *port*"という形式の文字列
persistpath	ターゲットプラットフォーム上の持続ファイルの絶対パス名または相対パス名

表 18 C_dfESPpubsubPersistModel の戻り値

値	説明
1	成功
0	失敗 — エラーはログに書き出されます。

C_dfESPpubsubQuiesceProject

この関数は、*projectURL* のエンジンに *projectURL* でプロジェクトを休止するよう指示します。この呼び出しは同期的です。つまり、プロジェクトが終了したときに戻ります。

int **C_dfESPpubsubQuiesceProject**(char **projectURL* <, clientObjPtr *client* >)

表 19 C_dfESPpubsubQuiesceProject のパラメータ

パラメータ	説明
projectURL	**"dfESP://*host* :*port*/ *project*"**という形式の文字列
client	C_dfESPsubscriberStart() 、C_dfESPpublisherStart()、“ C_dfESPGDsubscriberStart”、“ C_dfESPGDpublisherStart”のいずれかによって返されたクライアントオブジェクトへのオプションのポインタ。ヌルでない場合は、指定したクライアントのキューに入れられたイベントが処理されてから、プロジェクトを静止します。

表 20 C_dfESPpubsubQuiesceProject の戻り値

値	説明
1	成功
0	失敗 — エラーはログに書き出されます。

C_dfESPsubscriberMaxQueueSize

この関数を使用して、プロジェクト、クエリ、またはウィンドウ内のサブスクライバに送信されるイベントブロックをエンキューするために使用されるすべてのキューの最大サイズを構成します。これらのキューで消費されるメモリ量を制限するには、この値を使用します。block パラメータは、最大値に達したときの動作を指定します。

int **C_dfESPsubscriberMaxQueueSize**(char **serverURL*, int *maxSize* , int *block*)

表 21 C_dfESPsubscriberMaxQueueSize のパラメータ

パラメータ	説明
serverURL	次のいずれかの形式のサーバー URL の文字列。 ■ **"dfESP:// *host*:*port* "** ■ **"dfESP:// *host*:*port* /*project*"** ■ **"dfESP:// *host*:*port* /*project*/ *contquery*"** ■ **"dfESP:// *host*:*port* /*project*/ *contquery*/*window* "**
maxsize	*serverURL* 内のウィンドウへの単一のサブスクライバのためにキューに入れることができるイベントブロックの最大数
block	キューサイズが *maxSize* を下回るのを待つ場合は 1 に設定し、そうでない場合はクライアントを切断します

表 22 C_dfESPsubscriberMaxQueueSize の戻り値

値	説明
1	成功
0	失敗 — エラーはログに書き出されます。

C_dfESPpubsubSetPubsubLib

この関数は、クライアントと ESP サーバー間で使用されるトランスポートを変更します。

TCP/IP ピア以外を指定すると、ESP サーバーは対応するクライアントをアプライアンスに提供するために適切なコネクタを実行する必要があります。たとえば、クライアントのピアが Solace アプライアンスであると指定した場合、ESP サーバーは Solace コネクタを実行する必要があります。

アプライアンスで使用されるトピック名は、パブリッシュ/サブスクライブクライアントとコネクタによって調整され、アプライアンスを介してイベントブロックを正しくルーティングします。

注: この関数呼び出しはオプションです。これを呼び出すときは、C_dfESPpubsubInit()を呼び出す前に呼び出す必要があります。

int **C_dfESPpubsubSetPubsubLib**(C_dfESPpsLib *psLib*)

表 23 C_dfESPpubsubSetPubsubLib のパラメータ

パラメータ	説明
psLib	表 24 を参照してください

表 24 C_dfESPpubsubSetPubsubLib でサポートされている *psLib* の値

値	説明
ESP_PSLIB_NATIVE	ピアツーピアの TCP/IP ベースのソケット接続があります。これはデフォルト値です。
ESP_PSLIB_SOLACE	クライアントのピアは Solace アプライアンスです。アプライアンス接続パラメータを提供するために、**solace.cfg** という名前のクライアント設定ファイルがカレントディレクトリに存在する必要があります。または、C_dfESPpubsubSetTransportConfigFile() を呼び出して、クライアント設定ファイルを指定します。
ESP_PSLIB_RABBITMQ	クライアントのピアは Rabbit MQ サーバーです。Rabbit MQ サーバー接続パラメータを提供するために、**rabbitmq.cfg** という名前のクライアント設定ファイルがカレントディレクトリに存在する必要があります。または、C_dfESPpubsubSetTransportConfigFile() を呼び出して、クライアント設定ファイルを指定します。
ESP_PSLIB_KAFKA	クライアントのピアは Kafka クラスタです。Kafka クラスタ接続パラメータを提供するために、**kafka.cfg** という名前のクライアント設定ファイルがカレントディレクトリに存在する必要があります。または、C_dfESPpubsubSetTransportConfigFile() を呼び出して、クライアント設定ファイルを指定します。

表 25 トランスポートタイプ別の設定ファイル形式

トランスポートタイプ	設定ファイル形式
Solace	solace { SESSION_HOST = "10.37.150.244:55555" SESSION_USERNAME = "pub1"

トランスポートタイプ	設定ファイル形式

	SESSION_PASSWORD = "pub1"
	SESSION_VPN_NAME = "SAS"
	SESSION_RECONNECT_RETRIES = "5"
	SESSION_REAPPLY_SUBSCRIPTIONS = true
	SESSION_TOPIC_DISPATCH = true
	}
	sas
	{
	buspersistence = false
	queuename = "myqueue"
	protobuf = false
	protofile = "./GpbHistSimFactory.proto"
	protomsg = "GbpTrade"
	json = false
	dateformat = "%Y-%m-%d %H:%M:%S"
	passwordencrypted = false
	}
	注: passwordencrypted = true の場合、**SESSION_PASSWORD** の値は、この OpenSSL コマンドで生成された暗号化されたパスワードにする必要があります。echo "*SESSION_PASSWORD*" | openssl enc -e -aes-256-cbc -a -salt -pass pass:"espSOLclientUsedByUser= *SESSION_USERNAME*"

RabbitMQ	rabbitmq
	{
	host = "my.machine.com "
	port = "5672"
	exchange = "SAS"
	userid = "guest"
	password = "guest"
	passwordencrypted = false
	ssl = false
	sslcacert = "./mycacert.pem"
	sslkey = "./mykey.pem"
	sslcert = "./mycert.pem"
	heartbeat = "5"
	}
	
	sas
	{
	buspersistence = false
	queuename = "subpersist"
	protobuf = false
	protofile = "./GpbHistSimFactory.proto"
	protomsg = "GpbTrade"
	json = false
	noreplay = false
	noautoack = false
	dateformat = "%Y-%m-%d %H:%M:%S" (optional, default=null)
	}
	注: passwordencrypted = true の場合、password の値は、この OpenSSL コマンドで生成された暗号化されたパスワードにする必要があります。echo "*password*" | openssl enc -e -aes-256-cbc -a -salt -pass pass:"espRMQclientUsedByUser= *userid*"
	注: buspersistence と queuename パラメータは、Rabbit Mq メッセージのサブスクライバまたはパブリッシャにとって異なる意味を持ちます。パブリッシャの場合、queuename は常に無視されます。If buspersistence = false の場合、メッセージは非永続配信モードで送信されます。それ以外の場合、配信モードは永続的です。サブ

トランスポートタイプ	設定ファイル形式
	スクライバの場合は、受信キューを作成するときに常に queuename が使用されます。buspersistence = false の場合、クライアントによって作成されたすべてのキューおよびエクスチェンジは永続性がなく、自動的に削除されます。buspersistence = true の場合、すべての交換およびキューは永続性があり、自動削除ではありません。デフォルトでは、noreplay パラメータは false です。true に設定すると、Rabbit MQ から受信したメッセージは、buspersistence が有効になっていても肯定応答されます。デフォルトでは、noautoack パラメータは **false** に設定されています。**true** に設定すると、Rabbit MQ から受信したメッセージは、Rabbit MQ の autoack を通じて暗黙的に確認されるのではなく、明示的に確認されます。これは、受信したメッセージ処理で検出されたすべての ERROR が ack を抑制し、メッセージを Rabbit MQ キューに残すことを意味します。

Kafka	kafka
	{
	hostport = "kafkahost:9092"
	partition = "0"
	initialoffset = "largest"
	groupid = "mygroup"
	globalconfig = <librdkafka global config>
	topicconfig = <librdkafka topic config>
	metatopic= "mymetatopic" (optional, default=*random_string*)
	}
	
	sas
	{
	protobuf = false
	protofile = "./GpbHistSimFactory.proto"
	protomsg = "GpbTrade"
	json = false
	dateformat = "%Y-%m-%d %H:%M:%S"
	hotfailover = "false"
	numbufferedmsgs = "16000"
	zookeeperhostport = "myhost:myport"
	failovergroup = "mygroup"
	avro = false
	avroschemaregistryurl = "*myurl* "
	avroschemadefinition = "json_schema"
	avroschemaname = "*name*"
	}

注: Solace、Rabbit MQ、Kafka いずれかのトランスポートを使用する場合、次のパブリッシュ/サブスクライブ API 関数はサポートされていません。

- " C_dfESPpubsubGetModel"
- " C_dfESPGDpublisherStart"
- "C_dfESPGDpublisherGetID"
- " C_dfESPGDsubscriberStart"
- "C_dfESPGDsubscriberAck"
- "C_dfESPpubsubSetBufferSize"
- " C_dfESPsubscriberMaxQueueSize"

■ “C_dfESPpubsubPingHostPort”

C_dfESPpubsubSetTransportConfigFile

int **C_dfESPpubsubSetTransportConfigFile** (const char * *transportCfgFile*)

transportCfgFile パラメータは、トランスポート設定ファイルのフルパスを指定します。

表 26 C_dfESPpubsubSetTransportConfigFile の戻り値

値	説明
1	成功
0	失敗 — エラーはログに書き出されます。

C_dfESPdecodePubSubFailure

C_ESP_utf8str_t **C_dfESPdecodePubSubFailure** (C_dfESPpubsubFailures *failure*)

表 27 C_dfESPdecodePubSubFailure のパラメータ

パラメータ	説明
failure	C_dfESPpubsubFailures 列挙型

この関数は、C_dfESPpubsubFailures 列挙型のテキスト形式の説明を返します。

C_dfESPdecodePubSubFailureCode

C_ESP_utf8str_t **C_dfESPdecodePubSubFailureCode** (C_dfESPpubsubFailureCodes *code*)

表 28 C_dfESPdecodePubSubFailureCode のパラメータ

パラメータ	説明
code	C_dfESPpubsubFailureCodes 列挙型

この関数は、C_dfESPpubsubFailureCodes 列挙型のテキスト形式の説明を返します。

C_dfESPGDsubscriberAck

int **C_dfESPGDsubscriberAck**(clientObjPtr *client* , CdfESPeventblock *eventblock*)

表 29 C_dfESPGDsubscriberAck のパラメータ

パラメータ	説明
client	C_dfESPGDsubscriberStart() によって返されたクライアントオブジェクトへのポインタ
eventBlock	パブリッシャに返信されているイベントブロックへのポインタ。この関数が戻る前に、イベントブロックを解放してはいけません。

表 30 C_dfESPGDsubscriberAck の戻り値

値	説明
1	成功
0	失敗 — エラーはログに書き出されます。

C_dfESPGDpublisherCB_func

C_dfESPGDpublisherCB_func()

この関数は、パブリッシュされたイベントブロックの保証された配信ステータスをパブリッシャに返すために API によって呼び出されます。パラメータは、C_dfESPGDpublisherStart()に渡される保証配信パブリッシャコールバック関数のシグネチャです。

- パラメータ 1: READY または ACK または NACK (承認済みまたは未承認)。
- パラメータ 2: 64 ビットイベントブロック ID
- パラメータ 3: C_dfESPGDpublisherStart() に渡されたユーザコンテキストポインタ

void の場合は戻り値。

C_dfESPGDpublisherGetID

この関数は 64 ビットのイベントブロック ID を返します。イベントブロックに書き込まれる順番に固有の ID を取得するためにパブリッシャから呼び出され、保証配信対応のパブリッシュクライアントにそれらを注入することがあります。

C_dfESPGDpublisherGetID()

C_dfESPpubsubSetBufferSize

この関数は、ソケットの読み取りおよび書き込み操作に使用されるバッファのサイズを変更します。デフォルトでは、このサイズは 16MB です。

この関数呼び出しはオプションです。この関数を呼び出す場合、C_dfESPsubscriberStart() 、C_dfESPpublisherStart()、C_dfESPGDsubscriberStart()、C_dfESPGDpublisherStart()のいずれかの後、かつC_dfESPpubsubConnect()の前に呼び出す必要があります。

int **C_dfESPpubsubSetBufferSize**(clientObjPtr *client* , int32_t *mbytes*)

表 31 C_dfESPpubsubSetBufferSize のパラメータ

パラメータ	値
client	C_dfESPsubscriberStart() 、C_dfESPpublisherStart()、C_dfESPGDsubscriberStart()、C_dfESPGDpublisherStart()のいずれかによって返されたクライアントオブジェクトへのポインタ
mbytes	1MB 単位の読み書きバッファサイズ

表 32 C_dfESPpubsubSetBufferSize の戻り値

値	説明
1	成功
0	失敗 — エラーはログに書き出されます。

C_dfESPpubsubPingHostPort

この関数は、実行中の ESP サーバーに ping を実行して、指定されたポートが開いているかどうかを判断します。また、パブリッシュポートがパブリッシュ/サブスクライブポートであることを確認するために、マジックナンバーを交換して検証します。

int **C_dfESPpubsubPingHostPort**(char **serverURL*)

serverURL の場合、**"dfESP://*host*:*port*"**という形式の文字列を指定します。

表 33 C_dfESPpubsubSetBufferSize の戻り値

値	説明
1	ポートは開いており、パブリッシュ/サブスクライブポートです。
0	ポートが開いていないか、パブリッシュ/サブスクライブポートではありません。

C_dfESPpubsubSetTokenLocation

この関数は、パブリッシュ/サブスクライブサーバーによる認証に必要な OAuth トークンを含むローカルファイルシステム内のファイルの場所を設定します。

int **C_dfESPpubsubSetTokenLocation**(char * *tokenLocation*)

tokenLocation の場合、トークンを含むファイルのフルパスとファイル名を指定します。

表 34 C_dfESPpubsubSetTokenLocation の戻り値

値	説明
1	成功
0	失敗 — エラーはログに書き出されます。

Google プロトコルバッファオブジェクトをサポートするための機能

C_dfESPpubsubInitProtobuff

protobuffObjPtr **C_dfESPpubsubInitProtobuff** (char * *protoFile*, char * *msgName*, C_dfESPschema *C_schema* , char **dateFormat*, C_dfESPeventcodes *defaultOpcode*)

表 35 C_dfESPpubsubInitProtobuff のパラメータ

パラメータ	値
protoFile	メッセージ定義を含む Google の.proto ファイルへのパス。
msgName	Google .proto ファイル内のターゲットメッセージ定義の名前。
C_schema	ウィンドウスキーマへのポインタ。
dateFormat	CSV コンバージョンの日付形式。ESP_DATETIME および ESP_TIMESTAMP フィールド値を、エポック以降の秒数(ESP_DATETIME)またはマイクロ秒(ESP_TIMESTAMP)として解釈するには、ヌルに設定します。
defaultOpcode	シリアライズされた protobuf から構築されたイベントに挿入する演算コード。

この関数は、他のすべての protobuffAPI 関数に渡される、protobuff オブジェクトへのポインタを返します。失敗した場合はヌルを返します。

C_dfESPprotobuffToEb

C_dfESPeventblock **C_dfESPprotobuffToEb** (protobuffObjPtr *protobuff* , void * *serializedProtobuff*)

表 36 C_dfESPprotobuffToEb のパラメータ

パラメータ	説明
protobuff	"C_dfESPpubsubInitProtobuff"によって作成されたオブジェクトへのポインタ。
serializedProtobuff	ソケット上で受信されたシリアル化された protobuff へのポインタ

この関数はイベントブロックポインタを返します。

C_dfESPebToProtobuff

void ***C_dfESPebToProtobuff**(protobuffObjPtr *protobuff* , C_dfESPeventblock *C_eb*, int32_t *index*)

表 37 C_dfESPebToProtobuff のパラメータ

パラメータ	説明
protobuff	"C_dfESPpubsubInitProtobuff"によって作成されたオブジェクトへのポインタ。
C_eb	イベントブロックポインタ。
index	イベントブロックで変換するイベントのインデックス

この関数は、シリアル化された protobuf メッセージへのポインタを返します。

C_dfESPprotobuffToTrans

int **C_dfESPprotobuffToTrans**(protobuffObjPtr *protobuff* , void **serializedProtobuff*, C_dfESPeventV *trans*)

表 38 C_dfESPprotobuffToTrans のパラメータ

パラメータ	説明
protobuff	"C_dfESPpubsubInitProtobuff"によって作成されたオブジェクトへのポインタ。
serializedProtobuff	ソケットで受信したシリアル化された protobuf へのポインタ
trans	dfESPevent オブジェクトのポインタベクトルへのポインタ。

この関数は、Googleprotobuf メッセージをイベントポインタのベクトルへ変換します。

C_dfESPdestroyProtobuff

void **C_dfESPdestroyProtobuff**(protobuffObjPtr *protobuff* , void * *serializedProtobuff*)

表 39 C_dfESPdestroyProtobuff のパラメータ

パラメータ	説明
protobuff	"C_dfESPpubsubInitProtobuff"によって作成されたオブジェクトへのポインタ。
serializedProtobuff	C_dfESPPebToProtobuff()によって作成されたシリアル化された protobuff メッセージへのポインタ。

C_dfESPdeleteProtobuff

void **C_dfESPdeleteProtobuff**(protobuffObjPtr *protobuff*)

表 40 C_dfESPdeleteProtobuff のパラメータ

パラメータ	説明
protobuff	"C_dfESPpubsubInitProtobuff"によって作成されたオブジェクトへのポインタ。

この関数は、Googleprotobuf オブジェクトを削除します。

JSON オブジェクトをサポートする関数

C_dfESPpubsubInitJson

jsonObjPtr **C_dfESPpubsubInitJson**(C_dfESPschema *C_schema* , char * *dateFormat*, C_dfESPeventcodes *defaultOpcode* , char *separator*, uint32_t *doublePrecision*)

表 41 C_dfESPpubsubInitJson のパラメータ

パラメータ	説明
C_schema	ウィンドウスキーマへのポインタ。
dateFormat	CSV コンバージョンの日付形式。ESP_DATETIME および ESP_TIMESTAMP フィールド値を、エポック以降の秒数(ESP_DATETIME)

パラメータ	説明
	またはマイクロ秒(ESP_TIMESTAMP)として解釈するには、ヌルに設定します。
defaultOpcode	演算コードフィールドを含まない JSON から構築されたイベントに挿入する演算コード
separator	ソースウィンドウのフィールドでネストした JSON キーを区切る区切り文字。デフォルト値は_です。
doublePrecision	倍精度変数の ASCII 表現の小数点以下の桁数です。製品のデフォルト値を使用するには、値 6 を渡します。

この関数は、他のすべての JSON API 関数に渡される JSON オブジェクトへのポインタを返します。失敗した場合はヌルを返します。

C_dfESPjsonToEb

C_dfESPeventblock **C_dfESPjsonToEb**(jsonObjPtr *json* , const char **serializedJson*, int32_t *maxEvents*)

表 42 C_dfESPjsonToEb のパラメータ

パラメータ	説明
json	"C_dfESPpubsubInitJson"によって作成されたオブジェクトへのポインタ。
serializedJson	ソケットで受信されたシリアル化された JSON へのポインタ
maxEvents	作成されるイベントの数をこの値に制限します。値 0 は無制限を意味します。

この関数はイベントブロックポインタを返します。

C_dfESPebToJson

void ***C_dfESPebToJson**(jsonObjPtr *json* , C_dfESPeventblock *C_eb*)

表 43 C_dfESPebToJson のパラメータ

パラメータ	説明
json	"C_dfESPpubsubInitJson"によって作成されたオブジェクトへのポインタ。
C_eb	イベントブロックポインタ

シリアル化された JSON メッセージへのポインタを返します。

C_dfESPaddStaticJson

int **C_dfESPaddStaticJson**(jsonObjPtr *json* , char **staticJson*)

表 44 C_dfESPaddStaticJson のパラメータ

パラメータ	説明
json	“C_dfESPpubsubInitJson”によって作成されたオブジェクトへのポインタ。
staticJson	JSON テキストへのポインタ。

この関数は渡された JSON テキストを保存します。このテキストは、“C_dfESPjsonToEb”によって解析されるすべての JSON メッセージに追加されます。

C_dfESPjsonIgnoreMissingSchemaFields

void **C_dfESPjsonIgnoreMissingSchemaFields** (jsonObjPtr *json*)

表 45 C_dfESPjsonIgnoreMissingSchemaFields のパラメータ

パラメータ	説明
json	“C_dfESPpubsubInitJson”によって作成されたオブジェクトへのポインタ。

この関数は、“C_dfESPjsonToEb”を有効にして、ソースウィンドウスキーマに見つからない JSON キーを無視し、エラーを報告しません。

C_dfESPmatchSubstrings

void **C_dfESPmatchSubstrings**(jsonObjPtr *json* , char **substrings*)

表 46 C_dfESPmatchSubstrings のパラメータ

パラメータ	説明
json	“C_dfESPpubsubInitJson”によって作成されたオブジェクトへのポインタ。
substrings	カンマで区切られた *key*:*value* ペアを含む文字列。

“C_dfESPjsonToEb”はこの関数を使用して、JSON 入力への substrings パラメータで定義されている *key*: *value* のペアを比較します。キーは一致するが、値に *substrings* パラメータで定義されている値が含まれていない場合、*key*: *value* のペアは無視されます。

C_dfESPdestroyJson

void **C_dfESPdestroyJson**(jsonObjPtr *json* , void **serializedJson*)

表 47 C_dfESPdestroyJson のパラメータ

パラメータ	説明
json	“C_dfESPpubsubInitJson”によって作成されたオブジェクトへのポインタ。
serializedJson	シリアル化された JSON メッセージへのポインタ。

この関数は、“C_dfESPjsonToEb”によって作成されたシリアル化された JSON メッセージを破棄します。

C_dfESPdeleteJson

void **C_dfESPdeleteJson**(jsonObjPtr *json*)

表 48 C_dfESPdeleteJson のパラメータ

パラメータ	説明
json	“C_dfESPpubsubInitJson”によって作成されたオブジェクトへのポインタ。

この関数は JSON オブジェクトを削除します。

Apache Avro オブジェクトをサポートする関数

C_dfESPpubsubInitAvro

avroObjPtr **C_dfESPpubsubInitAvro**(C_dfESPschema *C_schema* , char * *dateFormat*, C_dfESPeventcodes *defaultOpcode* , char **schemaRegistryUrl*, char **schemaDef*, char **schemaName*, int32_t *isSub*, uint32_t *doublePrecision*)

表 49 C_dfESPpubsubInitAvro のパラメータ

パラメータ	説明
C_schema	ウィンドウスキーマへのポインタ。
dateFormat	CSV コンバージョンの日付形式。ESP_DATETIME および ESP_TIMESTAMP フィールド値を、エポック以降の秒数(ESP_DATETIME)またはマイクロ秒(ESP_TIMESTAMP)として解釈するには、ヌルに設定します。
defaultOpcode	演算コードフィールドを含まない Apache Avro メッセージから構築されたイベントに挿入する演算コード。
schemaRegistryUrl	Apache Avro スキーマレジストリの URL。
schemaDef	サブスクライバによって使用される Apache Avro スキーマ定義(JSON 形式)。このスキーマは起動時にスキーマレジストリにコピーされます。サブスクライブされたウィンドウスキーマと演算コードフィールドを表すスキーマを使用するには、ヌルに設定します。このフィールドもスキーマレジストリにコピーされます。
schemaName	スキーマレジストリからコピーし、サブスクライバによって使用される Apache Avro スキーマの名前。サブスクライブされたウィンドウスキーマと演算コードフィールドを表すスキーマを使用するには、ヌルに設定します。このフィールドもスキーマレジストリにコピーされます。
isSub	サブスクライバの場合は 1、パブリッシャの場合は 0 に設定します。
doublePrecision	倍精度変数の ASCII 表現の小数点以下の桁数です。製品のデフォルト値を使用するには、値 6 を渡します。

この関数は、他のすべての Apache Avro API 関数に渡される Apache Avro オブジェクトへのポインタを返します。失敗した場合はヌルを返します。

C_dfESPavroToEb

C_dfESPeventblock **C_dfESPavroToEb**(avroObjPtr *avro* , const char **serializedAvro*, int32_t *maxEvents*, size_t *length*)

表 50 C_dfESPavroToEb のパラメータ

パラメータ	説明
avro	“C_dfESPpubsubInitAvro”によって作成されたオブジェクトへのポインタ。
serializedAvro	ソケットで受信したシリアル化された Apache Avro メッセージへのポインタ。

パラメータ	説明
maxEvents	作成されるイベントの数をこの値に制限します。値 0 は無制限設定を意味します。
length	*serializedAvro* が指す Apache Avro メッセージの長さ。

この関数はイベントブロックポインタを返します。

C_dfESPebToAvro

void ***C_dfESPebToAvro**(avroObjPtr *avro* , C_dfESPeventblock *C_eb*)

表 51 C_dfESPebToAvro のパラメータ

パラメータ	説明
avro	"C_dfESPpubsubInitAvro"によって作成されたオブジェクトへのポインタ。
C_eb	イベントブロックポインタ。

この関数は、シリアル化された Apache Avro メッセージへのポインタを返します。

C_dfESPdestroyAvro

void **C_dfESPdestroyAvro**(avroObjPtr *avro* , void **serializedAvro*)

表 52 C_dfESPdestroyAvro のパラメータ

パラメータ	説明
avro	"C_dfESPpubsubInitAvro"によって作成されたオブジェクトへのポインタ。
serializedAvro	シリアル化された Apache Avro メッセージへのポインタ。

この関数は、"C_dfESPebToAvro"によって作成されたシリアル化された Apache Avro メッセージを破棄します。

C_dfESPdeleteAvro

void **C_dfESPdeleteAvro**(avroObjPtr *avro*)

表 53 C_dfESPdeleteAvro のパラメータ

パラメータ	説明
avro	"C_dfESPpubsubInitAvro"によって作成されたオブジェクトへのポインタ。

この関数は Apache Avro オブジェクトを削除します。

XML オブジェクトをサポートする関数

C_dfESPpubsubInitXml

xmlObjPtr **C_dfESPpubsubInitXml**(C_dfESPschema *C_schema* , char **dateFormat*, C_dfESPeventcodes *defaultOpcode* , uint32_t *doublePrecision*)

表 54 C_dfESPpubsubInitXml のパラメータ

パラメータ	説明
C_schema	ウィンドウスキーマへのポインタ
dateFormat	CSV コンバージョンの日付形式。ESP_DATETIME および ESP_TIMESTAMP フィールド値を、エポック以降の秒数(ESP_DATETIME)またはマイクロ秒(ESP_TIMESTAMP)として解釈するには、ヌルに設定します。
defaultOpcode	演算コードフィールドを含まない XML から構築されたイベントに挿入する演算コード
doublePrecision	倍精度変数の ASCII 表現の小数点以下の桁数です。製品のデフォルト値を使用するには、値 6 を渡します。

XML オブジェクトへのポインタを返します。このポインタは、他のすべての XML API 関数に渡されます。障害が発生した場合、値はヌルです。

C_dfESPxmlToEb

C_dfESPeventblockV **C_dfESPxmlToEb**(xmlObjPtr *xml* , const char **serializedXml*, int32_t *maxEvents*)

表 55 C_dfESPxmlToEb のパラメータ

パラメータ	説明
xml	"C_dfESPpubsubInitXml"によって作成されたオブジェクトへのポインタ。
serializedXml	ソケットで受信されたシリアル化された XML へのポインタ。
maxEvents	処理されるイベントの最大数。値 0 は無制限を意味します。

イベントブロックポインタを返します。

C_dfESPebToXml

void ***C_dfESPebToXml**(xmlObjPtr *xml* , C_dfESPeventblock *C_eb*)

表 56 C_dfESPebToXml のパラメータ

パラメータ	説明
xml	"C_dfESPpubsubInitXml"によって作成されたオブジェクトへのポインタ。
C_eb	イベントブロックポインタ。

シリアル化された XML メッセージへのポインタを返します。

C_dfESPaddStaticXml

int **C_dfESPaddStaticXml**(xmlObjPtr *xml* , char **staticXml*)

表 57 C_dfESPaddStaticXml のパラメータ

パラメータ	説明
xml	"C_dfESPpubsubInitXml"によって作成されたオブジェクトへのポインタ。
staticXML	XML テキストへのポインタ。

この関数は渡された XML テキストを保存します。このテキストは、" C_dfESPxmlToEb"によって解析されるすべての XML メッセージに追加されます。

C_dfESPxmlIgnoreMissingSchemaFields

void **C_dfESPxmlIgnoreMissingSchemaFields** (xmlObjPtr *xml*)

表 58 C_dfESPxmlIgnoreMissingSchemaFields のパラメータ

パラメータ	説明
xml	"C_dfESPpubsubInitXml"によって作成されたオブジェクトへのポインタ。

この関数により、"C_dfESPebToXml"はソースウィンドウスキーマに見つからない XML キーを無視し、エラーを報告しません。

C_dfESPdestroyXml

void **C_dfESPdestroyXml**(xmlObjPtr *xml* , void **serializedXml*)

表 59 C_dfESPdestroyXml のパラメータ

パラメータ	説明
json	"C_dfESPpubsubInitXml"によって作成されたオブジェクトへのポインタ。
serializedXml	シリアル化された XML メッセージへのポインタ。

この関数は、"C_dfESPebToXml"によって作成されたシリアル化された XML メッセージを破棄します。

C_dfESPdeleteXml

void **C_dfESPdeleteXml**(xmlObjPtr *xml*)

表 60 C_dfESPdeleteXml のパラメータ

パラメータ	説明
xml	"C_dfESPpubsubInitXml"によって作成されたオブジェクトへのポインタ。

この関数は XML オブジェクトを削除します。

イベントストリーム処理オブジェクトを分析および操作する関数

AC ライブラリには、クライアント開発者がアプリケーションまたはサーバーからのイベントストリーム処理オブジェクトを分析および操作できるようにする一連の関数が用意されています。これらの関数は、C++ Modeling API で提供されるメソッドの小さなサブセットを囲む C ラッパーのセットで

す。これらのラッパーを使用すると、クライアント開発者は C++ではなく C を使用できます。これらのオブジェクトの例は、イベント、イベントブロック、およびスキーマです。これらの呼び出しのサブセットは、次のとおりです。

完全な呼び出しについては、**$DFESP HOME/doc/html** にある API リファレンスのドキュメントを参照してください。

表 61 イベントストリーム処理オブジェクトを操作する関数のサブセット

リクエストしたいアクション	関数
イベントブロックのサイズを取得します	C_ESP_int32_t eventCnt = C_dfESPeventblock_getSize(eb);
イベントブロックからイベントを抽出します	C_dfESPevent ev = C_dfESPeventblock_getEvent(eb, eventIndx);
オブジェクト(この場合はスキーマの文字列表現)を作成します	C_ESP_utf8str_t schemaCSV = C_dfESPschema_serialize(schema);
オブジェクト(この場合は文字列のベクトル)を解放します	C_dfESPstringV_free(metaVector);

Google プロトコルバッファのパブリッシュ/サブスクライブ API サポート

Google プロトコルバッファのパブリッシュ/サブスクライブ API サポートの概要

SAS Event Stream Processing では、Google プロトコルバッファをサポートするライブラリが提供されます。このライブラリは、バイナリ形式のイベントブロックとシリアル化された Google プロトコルバッファ（protobuf)の間の変換メソッドを提供します。

protobuf をイベントストリーム処理サーバーと交換するために、標準パブリッシュ/サブスクライブ API を使用するパブリッシュ/サブスクライブクライアントは、C_dfESPpubsubInitProtobuff()を呼び出して、Google プロトコルバッファをサポートするライブラリをロードできます。次に、protobuf 形式のソースデータを持つパブリッシャクライアントは、C_dfESPprotobuffToEb()を呼び出して、C_dfESPpublisherInject()を呼び出す前にバイナリ形式のイベントブロックを作成できます。同様に、サブスクライバクライアントは、C_dfESPebToProtobuff()を呼び出して protobuf ハンドラに渡す前に、受け取ったイベントブロックを protobuf に変換することができます。

注: イベントストリーム処理パブリッシュ/サブスクライブ接続のサーバー側は、Google プロトコルバッファをサポートしていません。イベントブロックはバイナリ形式でのみ送受信し続けます。

SAS Event Stream Processing Java パブリッシュ/サブスクライブ API には、Java パブリッシュ/サブスクライブクライアント用の同等のメソッドを実装する protobuf JAR ファイルが含まれています。

パブリッシュ/サブスクライブクライアント接続は、そのウィンドウのスキーマを使用して 1 つのウィンドウとイベントを交換します。したがって、protobuf 対応のクライアント接続では、**.proto** ファイルのメッセージブロックで定義されているように、単一の protobuf 固定メッセージ型を使用します。Google プロトコルバッファをサポートしているライブラリは、メッセージ定義を動的に解析するので、あらかじめコンパイルされたメッセージ固有のクラスは必要ありません。ただし、Java ライブラリは**.proto** ファイルの代わりに**.desc** ファイルを使用します。このため、対応する**.desc** ファイルを生成するために**.proto** ファイルで Google **protoc** コンパイラを実行する必要があります。

C クライアントの場合、**.proto** ファイルの名前と囲まれたメッセージは、両方とも C_dfESPpubsubInitProtobuff()呼び出しで Google プロトコルバッファをサポートするライブラリに渡されます。この呼び出しは、protobuf オブジェクトインスタンスを返します。このインスタンスは、その後、クライアントによってすべての 後続の protobuf 呼び出しに渡されます。このインスタンスは protobuf メッセージ定義に固有です。したがって、特定のウィンドウへのクライアント接続がアップしている限り有効です。クライアントが停止して再起動すると、新しい protobuf オブジェクトインスタンスを取得する必要があります。

Java クライアントの場合、プロセスは少し異なります。クライアントは dfESPprotobuf オブジェクトのインスタンスを作成し、その init()メソッドを呼び出します。その後の protobuf 呼び出しは、このオブジェクトのメソッドを使用して行われ、C++ protobuf オブジェクトで説明されているのと同じ有効範囲が適用されます。

バイナリイベントブロックと protobuf との間の変換は、protobuf メッセージ定義内のフィールドを、関連するパブリッシュ/サブスクライブウィンドウのスキーマ内のフィールドに一致させることによって行われます。protobuf メッセージ定義とウィンドウスキーマが互換性があることを確認してください。protobuf メッセージ定義にオプションのフィールドが含まれている場合は、それらがウィンドウスキーマに含まれていることを確認してください。受信した protobuf メッセージにオプションのフィールドがない場合、イベントの対応するフィールドはヌルに設定されます。逆に、protobuf を構築し、イベントのフィールドにヌルが含まれている場合、対応する protobuf フィールドは設定されていないため、**.proto** ファイルでオプションとして定義する必要があります。

次のデータ型マッピングはサポートされていません。

表 62 イベントストリームプロセッサのデータ型から Google Protocol Buffer のデータ型へのマッピング

イベントストリームプロセッサのデータ型	Google プロトコルバッファデータタイプ
BINARY	TYPE_BYTES
DATE	TYPE_INT64
DOUBLE	TYPE_DOUBLE TYPE_FLOAT
INT32	TYPE_INT32 TYPE_UINT32 TYPE_FIXED32 TYPE_SFIXED32 TYPE_SINT32

イベントストリームプロセッサのデータ型	Google プロトコルバッファデータタイプ
	TYPE_ENUM
INT64	TYPE_INT64 TYPE_UINT64 TYPE_FIXED64 TYPE_SFIXED64 TYPE_SINT64
STAMP	TYPE_INT64
STRING	TYPE_STRING

重要 次のデータ型はサポートされていません。

- RSTRING
- ARRAY(*)

プロトコルバッファメッセージ内のネストされたフィールドと繰り返しフィールドをイベントブロックに変換する

それらがサポートされている場合は、protobuf のメッセージフィールドを繰り返すことができます。TYPE_MESSAGE のメッセージフィールドはネストすることができ、場合によっては繰り返すこともできます。protobuf メッセージをイベントブロックに変換する際には、次のポリシーに従って、これらのすべてのケースがサポートされます。

- ネストされたメッセージを含む protobuf メッセージは、対応するスキーマが protobuf メッセージの平坦な表現であることを必要とします。たとえば、最初のメッセージが 4 つのフィールドを持つネストされたメッセージで、2 つ目がネストされていない、3 つ目が 2 つのフィールドを持つネストされたメッセージである 3 つのフィールドを含む protobuf メッセージは、4 + 1 + 2 = 7 フィールドのスキーマを必要とします。ネストの深さは制限されません。
- 単一の protobuf メッセージは、ネストされたメッセージフィールドが繰り返されない限り、単一のイベントを含むイベントブロックに常に変換されます。
- protobuf メッセージが繰り返される非メッセージタイプのフィールドを持つ場合、そのフィールドのすべてのエレメントは、イベントのカンマで区切られた単一の文字列フィールドに集められます。このため、protobuf メッセージの繰り返しフィールドに対応するスキーマフィールドは、protobuf メッセージのフィールドタイプに関係なく、タイプ ESP_UTF8STR でなければなりません。
- 繰り返されるネストされたメッセージフィールドを含む protobuf メッセージは、常に複数のイベントを含むイベントブロックに変換されます。反復されるすべてのネストされたメッセージフィールド内の各エレメントに対して 1 つのイベントがあります。

イベントブロックをプロトコルバッファメッセージに変換

イベントブロックを protobuf に変換することは、protobuf 内のネストされたフィールドと繰り返しフィールドをイベントブロックに変換することと概念的には似ていますが、プロセスにはより多くのコードが必要です。イベントブロック内のすべてのイベントは、別々の protobuf メッセージに変換されます。このため、C_dfESPebToProtobuff()ライブラリ呼び出しは、イベント・ブロック内のどのイベントを変換するかを示す索引パラメーターをとります。イベントブロック内のすべてのイベントに対して、ライブラリをループで呼び出す必要があります。

変換により、生成された protobuf メッセージのネストされたフィールドが正しくロードされます。イベントブロックは繰り返しフィールドをサポートしていないため、結果の protobuf メッセージの繰り返しフィールドには 1 つのエレメントしか含まれていません。

注: protobuf 変換へのイベントブロックは、イベント演算コードが protobuf メッセージにコピーされないため、Insert 演算コードを持つイベントのみをサポートします。protobuf からイベントブロックへの変換では、C C_dfESPpubsubInitProtobuff()関数または Java の init()関数で指定された演算コードを使用します。protobufs がコネクタまたはアダプタによって呼び出される場合、演算コードは、コネクタまたはアダプタが Upsert を使用するように構成されていない限り Insert です。

Google プロトコルバッファの転送のサポート

次のメッセージバスに関連付けられているコネクタとアダプタを使用すると、Google プロトコルバッファのサポートが利用できます。

- IBM WebSphere MQ
- Rabbit MQ
- Solace
- Tibco/RV
- Kafka

これらのコネクタとアダプタは、バイナリイベントブロックではなく、メッセージバスを介して protobuf の転送をサポートします。これにより、パブリッシュ/サブスクライブ API を使用せずに、サードパーティのパブリッシャまたはサブスクライバがメッセージバスに 接続し、エンジンと protobuf を交換することができます。protobuf メッセージ形式とウィンドウスキーマは互換性がなければなりません。

コネクタまたはアダプタは、**protofile** および **protomsg** パラメータを介して**.proto** ファイルおよびメッセージ名の構成を必要とします。コネクタは、Google プロトコルバッファをサポートする SAS Event Stream Processing ライブラリを使用して、protobuf をイベントブロックとの間で変換します。さらに、C および Java Solace パブリッシュ/サブスクライブクライアントは、**solace.cfg** クライアント構成ファイルで構成されている場合、Google プロトコルバッファもサポートします。同様に、**rabbitmq.cfg** または **kafka.cfg** クライアント構成ファイルで C RabbitMQ および Kafka パブリッシュ/サブスクライブクライアントが Google プロトコルバッファをサポートするように構成されている場合、C RabbitMQ および Kafka パブリッシュ/サブスクライバクライアントもこれらをサポートします。protobuf 対応のクライアントパブリッシャは、イベントブロックを protobuf に変換し、メッセ

ージバスを介して、Google プロトコルバッファのサードパーティコンシューマーに転送します。同様に、protobuf 対応のクライアントサブスクライバは、メッセージバスから protobuf を受信し、それをイベントブロックに変換します。

JSON メッセージングのパブリッシュ/サブスクライブ API サポート

概要

SAS Event Stream Processing では、JSON メッセージングをサポートする C ライブラリが提供されます。ライブラリは、バイナリ形式のイベントブロックとシリアライズされた JSON メッセージの間の変換メソッドを提供します。

JSON メッセージをイベントストリーム処理サーバーと交換するには、標準パブリッシュ/サブスクライブ API を使用するパブリッシュ/サブスクライブクライアントは、C_dfESPpubsubInitJson()を呼び出して JSON をサポートするライブラリをロードできます。JSON 形式のソースデータを持つパブリッシャクライアントは、C_dfESPjsonToEb()を呼び出して、バイナリ形式のイベントブロックを作成できます。その後、C_dfESPpublisherInject()を呼び出すことができます。

同様に、サブスクライバクライアントは、受け取ったイベントブロックを JSON メッセージハンドラに渡す前に C_dfESPebToJson()を呼び出して JSON メッセージに変換できます。

注: パブリッシュ/サブスクライブ接続を処理するイベントストリームのサーバー側では、JSON メッセージはサポートされません。イベントブロックはバイナリ形式でのみ送受信し続けます。

パブリッシュ/サブスクライブクライアント接続は、そのウィンドウのスキーマを使用して 1 つのウィンドウとイベントを交換します。これに対応して、JSON 対応のクライアント接続は、固定された JSON スキーマとメッセージを交換します。ただし、このスキーマの静的な定義はありません。JSON メッセージと関連ウィンドウスキーマの間のスキーマの不一致は、実行時にのみ検出されます。

C_dfESPpubsubInitJson()コールは、JSON オブジェクトインスタンスを返します。このインスタンスは、その後のすべての JSON 呼び出しでクライアントによって 渡されます。このオブジェクトインスタンスは、特定のウィンドウへのクライアント接続が起動している間のみ有効です。クライアントが停止して Restart すると、新しい JSON オブジェクトインスタンスを取得する必要があります。

基本的に、単一の JSON メッセージは単一のイベントにマッピングされます。ただし、JSON メッセージ内に複数のイベントを含めることができます。

JSON メッセージのネストされたフィールドをイベントブロックに変換する

ウィンドウスキーマは、JSON イベントスキーマのフラット化された表現でなければなりません。ここで、ウィンドウフィールド名は、ネストされた JSON タグ名の連結をアンダースコアで区切ったものです。入力 JSON に、ソースウィンドウスキーマに意図的に欠けているフィールドがあり、無視す

る必要がある場合は、イベントブロックを作成する前に C_dfESPignoreMissingSchemaFields()メソッドを呼び出します。これにより、ライブラリが ERROR をログに記録することが防止されます。

たとえば、バイトオーダーのエンディアンが異なる場合でも、 JSON イベントスキーマ内では、配列とオブジェクトの無制限のネストがサポートされています。

JSON イベントに配列フィールドが含まれている場合、対応する ESP ウィンドウフィールドのタイプは次のとおりです。

- ESP_UTF8STR: すべての JSON タイプをサポートしています。これは、JSON 配列を、配列の値の文字列バージョンを含む 1 つのコンマ区切りの文字列に変換します。
- ESP_ARRAY_I32: JSON の Boolean 配列のみをサポートしています。
- ESP_ARRAY_I64: JSON の整数配列と JSON のブール配列をサポートしています。
- ESP_ARRAY_DBL: JSON の実在する配列をサポートします。

JSON イベントから構築されたイベントには常に Insert 演算コードがあります。例外は、JSON イベントに opcode という名前のフィールドが含まれている場合です。その場合、そのフィールドの値はイベント演算コードを設定するために使用されます。他の JSON フィールドがソースウィンドウスキーマのフィールドと一致しない場合、注入操作は失敗します。

表 63 JSON イベントの演算コードフィールドの有効な値

値	演算コード
i **I**	Insert
u **U**	Update
d **D**	Delete
p **P**	Upsert
s **S**	Safe delete

デフォルトでは、イベントブロックは type= normal です。イベントブロック配列内のイベントが追加の配列内に含まれている場合(つまり、追加のかっこで囲まれている場合)、イベントブロックは type= transactional。

設定された部分文字列を含まない JSON 値を除外するように JSON ライブラリを設定することができます。この場合、対応するイベントは静かにドロップされます。このようなフィルタリングを有効にするには、matchsubstrings パラメータを設定し、カンマで区切られた ***key*:*substring*** ペアの文字列を指定します。たとえば、"foo:bar"その値に "bar"を含まない JSON キー "foo"がイベントストリーム処理イベントを生成しないことを意味します。キーごとに複数の部分文字列を設定する場合、入力 JSON データには、イベントを生成するために構成された部分文字列のうちの 1 つのみが含まれている必要があります。

JSON フィールド(またはフィールドの組み合わせ)をソースウィンドウのキーとして簡単に使用できない場合は、次のキーフィールドのいずれかまたは両方をソースウィンドウスキーマに追加します。

"*eventindex:int64,*adapterindex**:*文字列*"**

ソースウィンドウスキーマに存在する場合、ライブラリは次のようにこれらのフィールドに入力します。

- *eventindex*: イベントごとにインクリメントされる 64 ビットの整数値。プロセス空間内の複数のライブラリにまたがって一意であることが保証されています。
- *adapterindex*: 異なるプロセス空間で実行されているライブラリのインスタンスに対して一意の GUID 文字列。

これらの値を組み合わせることにより、異なるプロセスのライブラリの複数のユーザーが、キーを複製することなくイベントを単一の Source ウィンドウに挿入できます。

注入されたすべてのイベントに静的な JSON 値を含める必要がある場合は、C_dfESPaddStaticJson() メソッドに渡します。後続のすべてのイベントには、その JSON から構築されたフィールドが含まれます。

イベントブロックを JSON メッセージに変換する

イベントから作成されたすべての JSON イベントには、opcode フィールドが含まれています。有効な値は、表 63 にリストされています。

入力はイベントブロックなので、結果として得られる JSON メッセージは常にルートオブジェクトとして配列を持ちます。各配列エントリは単一のイベントを表します。

データ変数型 STAMP、DATE および MONEY は、フィールド値の CSV 表現を含む JSON 文字列に変換されます。

JSON メッセージの転送のサポート

JSON メッセージングのサポートは、次のメッセージバスに関連付けられたコネクタとアダプタを使用すると利用できます。

- IBM WebSphere MQ
- RabbitMQ
- Solace
- Tibco/RV
- Kafka

これらのコネクタとアダプタは、バイナリイベントブロックではなく、メッセージバスを介して JSON エンコードされたメッセージの転送をサポートします。これにより、サードパーティのパブリッシャまたはサブスクライバは、パブリッシュ/サブスクライブ API を使用せずに、メッセージバスに接続し、JSON メッセージをエンジンと交換できます。

メッセージ形式構成は必要ありません。JSON スキーマとウィンドウスキーマが互換性がない場合、JSON をイベントブロックに変換するパブリッシャは、イベントブロックが挿入されると失敗します。コネクタは、JSON 変換をサポートする SAS Event Stream Processing ライブラリを使用して、JSON をイベントブロックとの間で変換します。

RabbitMQ、Solace Systems、および Kafka パブリッシュ/サブスクライブクライアントは、**rabbitmq.cfg**、**solace.cfg**、または **kafka.cfg** クライアント設定ファイルで JSON をサポートするように設定されている場合、JSON をサポートします。JSON 対応クライアントパブリッシャは、イベントブロックを JSON メッセージに変換して、メッセージバスを介して、JSON メッセージのサードパーテ

ィコンシューマーに転送します。同様に、JSON 対応クライアントのサブスクライバは、メッセージバスから JSON メッセージを受け取り、それをイベントブロックに変換します。

Apache Avro メッセージのパブリッシュ/サブスクライブ API サポート

概要

SAS Event Stream Processing は、Apache Avro メッセージをサポートするための C ライブラリを提供します。このライブラリは、バイナリ形式のイベントブロックをシリアル化された Apache Avro メッセージに変換するメソッドと、シリアル化された Apache Avro メッセージをバイナリ形式のイベントブロックに変換するメソッドを提供します。このライブラリは Linux では利用できますが、Windows プラットフォームでは利用できません。スキーマレジストリが必要です。libserdes でサポートされている任意のシリアル化形式がサポートされています。

Apache Avro メッセージをイベントストリーム処理サーバーと交換するには、標準パブリッシュ/サブスクライブ API を使用するパブリッシュ/サブスクライブクライアントは、C_dfESPpubsubInitAvro()を呼び出して Apache Avro をサポートするライブラリをロードできます。次に、Apache Avro フォーマットのソースデータを持つパブリッシャクライアントは C_dfESPavroToEb()を呼び出してバイナリ形式でイベントブロックを作成し、C_dfESPpublisherInject()を呼び出してパブリッシュできます。

同様に、サブスクライバクライアントは、受け取ったイベントブロックを Apache Avro メッセージハンドラに渡す前に、C_dfESPebToAvro()を呼び出して Apache Avro メッセージに変換できます。

注: パブリッシュ/サブスクライブ接続を処理するイベントストリームのサーバー側では、Apache Avro メッセージはサポートされません。イベントブロックはバイナリ形式でのみ送受信し続けます。

パブリッシュ/サブスクライブクライアント接続は、そのウィンドウのスキーマを使用して 1 つのウィンドウとイベントを交換します。しかし、ApacheAvro メッセージには可変スキーマが含まれているため、Apache Avro 対応のクライアント接続はプロジェクションベースでメッセージを交換します。Apache Avro スキーマとウィンドウスキーマの間で共通のフィールドが転送されます。その他は NULL またはデフォルト値のままです。一致しなかったフィールドごとに対応するログメッセージが生成されます。

C_dfESPpubsubInitAvro()呼び出しは、Apache Avro オブジェクトインスタンスを返します。その後、このインスタンスはクライアントによって 、その後のすべての Apache Avro 呼び出しで渡されます。このインスタンスは、特定のウィンドウへのクライアント接続が起動している間のみ有効です。クライアントが停止して再起動すると、新しい Apache Avro オブジェクトインスタンスを取得する必要があります。

Apache Avro メッセージのイベントブロックへの変換

ウィンドウスキーマは、Apache Avro メッセージスキーマ内の 1 つ以上のフィールドをフラットに表現したものでなければなりません。ウィンドウフィールド名は、ネストされた Apache Avro フィールド名をアンダースコアで区切って連結したものです。Apache Avro スキーマの JSON 表現を使用して、ウィンドウスキーマのフィールド名を派生させます。

必要に応じて、パブリッシャの"debug"レベルのログ記録を有効にして、受信したメッセージごとに Apache Avro スキーマの JSON 表現を記録できます。

ウィンドウフィールド名を作成するには、name キーのネストされた値を連結します。複合型(record、enum、および fixed)が共用体の一部として含まれる場合は、それらの名前のみを含めます。レコード全体のルート名を含めないでください。

たとえば、次のスキーマを考えてみましょう。

```
{
 "type": "record",
 "name": "EnhancedEvent",
 "namespace": "com.sas.mkt.kafka.domain.events",
 "fields": [
  {
   "name": "impression",
   "type": [
    "null",
    {
     "type": "record",
     "name": "Impression",
     "namespace": "com.sas.mkt.kafka.domain.common",
     "fields": [
      {
       "name": "controlGroup",
       "type": [
        "null",
        {
         "type": "string",
         "avro.java.string": "String"
        }
       ],
       "default": null
      },
      {
       "name": "creativeId",
       "type": [
        "null",
        {
         "type": "string",
         "avro.java.string": "String"
        }
       ],
       "default": null
      },
```

...

creativeId キーに対応するウィンドウスキーマフィールド名は次のとおりです。

impression_Impression_creativeId

Apache Avro メッセージスキーマ内では、無制限のネストがサポートされています。

Apache Avro スキーマに配列フィールドが含まれている場合、その配列内のすべてのエレメントは、イベント内のカンマで区切られた単一の文字列フィールドにまとめられます。このため、Apache Avro メッセージの配列フィールドに対応するスキーマフィールドのタイプは、メッセージスキーマ内のフィールドタイプに関係なく ESP_UTF8STR でなければなりません。

Apache Avro ライブラリが別のデフォルトの演算コードで初期化されていない限り、Apache Avro メッセージから作成されたイベントには常に Insert 演算コードがあります。この初期化は、Kafka パブリッシャコネクタまたはアダプタ上で publishwithupsert パラメーターを設定することで実現できます。Apache Avro メッセージに opcode という名前のフィールドが含まれている場合は例外です。その場合、そのフィールドの値はイベント演算コードを設定するために使用されます。

表 64 Apache Avro イベントの演算コードフィールドの有効な値

値	演算コード
i **I**	Insert
u **U**	Update
d **D**	Delete
p **P**	Upsert
s **S**	Safe delete

イベントブロックには type= normal があります。Apache Avro フィールド(またはフィールドの組み合わせ)をソースウィンドウのキーとして簡単に使用できない場合は、次のキーフィールドをソースウィンドウスキーマに追加します。

"eventindex*:int64"

ソースウィンドウスキーマに存在する場合、ライブラリは次のようにこのフィールドに入力します。

表 65 ソースウィンドウスキーマのキーフィールド

フィールド	説明
eventindex	イベントごとに増分される 64 ビットの原子整数。プロセス空間内のライブラリの複数のインスタンスにわたって一意であることが保証されています。

イベントブロックの Avro メッセージへの変換

イベントブロックから Apache Avro メッセージを作成する場合、含まれる Apache Avro スキーマは次のいずれかです。

- サブスクライブされたウィンドウスキーマ。これはデフォルト値で、デフォルトで opcode フィールドが含まれています。(有効な値は前のテーブルにリストされています。)起動時に、このスキーマは構成済みの ApacheAvro スキーマレジストリに登録されます。デフォルトでは、スキーマの登録に使用される名前は **window_schema** です。C_dfESPpubsubInitAvro()関数のパラメータ schemaName が null でない場合、スキーマはそのパラメータの値を使用して登録されます。
- null でない場合、Apache Avro スキーマは C_dfESPpubsubInitAvro()関数の schemaDef パラメータに渡されます。開始時に、このスキーマは、C_dfESPpubsubInitAvro()関数の schemaName パラメータで渡される名前を使用して、構成済みの Apache Avro スキーマレジストリに登録されます。
- Apache Avro スキーマは C_dfESPpubsubInitAvro()関数の schemaName パラメータで渡される名前を使用して、設定済みの Apache Avro スキーマレジストリから取得されます。

イベントブロック内のすべてのイベントに対して 1 つの Apache Avro メッセージが生成されます。

データ変数型 **MONEY** はサポートされていません。ターゲットの Apache Avro タイプに応じて、データ変数型 **STAMP** と **DATE** は次のいずれかです。

- 64 ビット整数としてコピー
- フィールド値の CSV 表現を含む文字列に変換

Avro メッセージの転送のサポート

Kafka コネクタまたはアダプタ、Kinesis コネクタまたはアダプタ、あるいは Kafka トランスポートに対応している別の C アダプタを使用する場合、Apache Avro メッセージがサポートされます。これらのアダプタは、ESP サーバーへの直接接続ではなく、メッセージバスを介した Apache Avro エンコードメッセージの転送をサポートします。このサポートにより、サードパーティのパブリッシャまたはサブスクライバは、パブリッシュ/サブスクライブ API を使用せずに、メッセージバスに接続し、Apache Avro メッセージをエンジンと交換できます。メッセージ形式構成は必要ありません。これらのアダプタは、前述の Apache Avro 変換をサポートする SAS Event Stream Processing ライブラリを使用して、Apache Avro とイベントブロック間の変換を行います。

Kafka アダプタまたはコネクタは、コネクタまたはアダプタ設定で Apache Avro 変換が有効になっています。詳細については、*“Kafka コネクタとアダプターの使用” (SAS Event Stream Processing: コネクタとアダプター)*を参照してください。

Kinesis アダプタまたはコネクタは、コネクタまたはアダプタ設定で Apache Avro 変換が有効になっています。詳細については、*“Kinesis コネクタとアダプターの使用” (SAS Event Stream Processing: コネクタとアダプター)*を参照してください。

Kafka トランスポートを使用するその他の C アダプタは、**kafka.cfg** クライアントトランスポート設定ファイルで Apache Avro 変換が有効になっています。

XML メッセージングのパブリッシュ/サブスクライブ API サポート

SAS Event Stream Processing では、XML メッセージングをサポートする C ライブラリが提供されます。このライブラリは、以前に説明した JSON メッセージングライブラリに似ています。次の API 関数を提供します。

- C_dfESPpubsubInitXml()
- C_dfESPxmlToEb()
- C_dfESPebToXml()
- C_dfESPaddStaticXml()
- C_dfESPignoreMissingSchemaFields()

これらの関数は、JSON ライブラリの対応する JSON 関数と同じ操作を実行します。

重要 次のデータ型はサポートされていません。

- RSTRING
- BLOB (バイナリラージオブジェクト)
- ARRAY(*)

保証された配信

保証された配信の概要

C、Java、および Python パブリッシュ/サブスクライブ API は、単一のパブリッシャと複数のサブスクライバ間の保証された配信をサポートします。保証配信では、ソースウィンドウにパブリッシュされる各イベントブロックが、サブスクライブされたウィンドウ内に正確に 1 つのイベントブロックを生成するモデルが想定されています。この 1 ブロックのブロックアウト原則は、パブリッシュされたすべてのイベントブロックに対して保持する必要があります。保証配信確認メカニズムは、モデルによって実行されるイベント処理を認識していません。

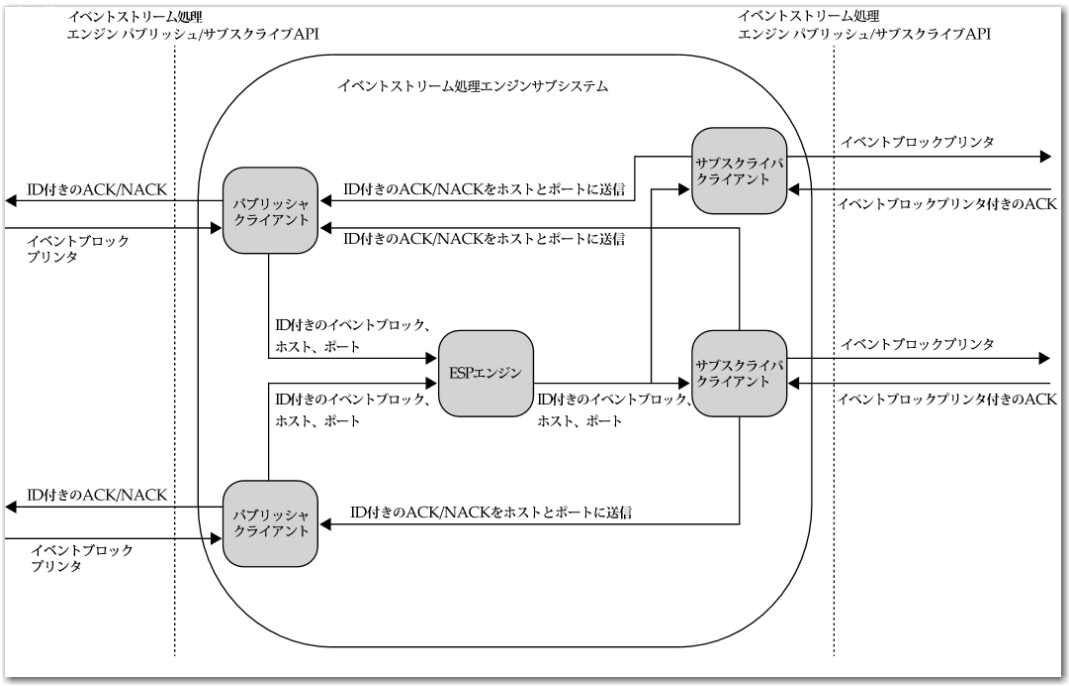
パブリッシュまたはサブスクライブ接続が開始されると、さまざまなパブリッシュ/サブスクライブ・アクティビティを 実行するためにクライアントが確立されます。パブリッシュ接続が開始されると、そのイベントブロックの配信を確認するのに必要な保証されたサブスクライバの数が指定されます。予想されるすべてのサブスクライブからの非受信時に否定応答を生成するために使用されるタイムアウト値も指定されます。パブリッシャによって挿入されたすべてのイベントブロックには、パブリッ

シャによって設定された一意の 64 ビット ID が含まれています。この ID は、パブリッシャのユーザー定義のコールバック関数で、すべての肯定応答または否定応答でパブリッシュクライアントからパブリッシャに返されます。この関数は、パブリッシュクライアントが開始されたときに登録されます。

サブスクライブ・コネクションが開始されると、サブスクライブ・クライアントは、一連の保証された配信パブリッシャをホストおよびポート・エントリのリストとして渡します。次に、クライアントは、リスト上の各パブリッシャとの TCP 接続を確立します。この接続は、このパブリッシャ/サブスクライバのペアに固有の肯定応答を転送するためにのみ使用されます。サブスクライバは、新しいパブリッシュ/サブスクライブ API 関数を呼び出して肯定応答をトリガします。

イベントブロックには新しいホスト、ポート、および ID フィールドが含まれています。すべてのイベントブロックは、これらのフィールドの組み合わせによって一意に識別されます。これにより、サブスクライバは重複(つまり再送信)イベントブロックを識別できます。

図 1 保証配信データフロー図



次の点に注意してください:

- 保証配信が可能な API 関数を使用しないパブリッシャおよびサブスクライバは、暗黙のうちに配信が無効になっていることを保証します。
- 保証配信サブスクライバは、保証されていない配信サブスクライバと混合することができます。
- 保証配信可能パブリッシャは、READY コールバックが受信されるまでパブリッシュを開始するのを待機する可能性があります。これは、設定されたサブスクライバ数がパブリッシャに肯定応答接続を確立したことを示します。
- エンジンによって生成されたスナップショットの結果として保証配信可能なサブスクライバによって受信されたイベントブロックは承認されません。
- 特定の条件下では、サブスクライバは重複したイベントブロックを受信します。これらの条件には、次が含まれます。
  - 関連するすべてのサブスクライバが開始する前に、パブリッシャがパブリッシュを開始します。開始されたサブスクライバは、開始されたサブスクライバの数がパブリッシャが渡す必要な肯定応答の数に達するまで、重複するイベントブロックを受信できます。
  - 配信可能な保証付きサブスクライバは、パブリッシャがパブリッシュする間に切断されます。これにより、以前に説明したのと同じシナリオがトリガーされます。

  - 遅いサブスクライバは、イベントブロックのタイムアウトを引き起こし、パブリッシャへの否定応答をトリガします。この場合、パブリッシャに関連するすべてのサブスクライバは、それらのブロックに対してすでに C_dfESPGDsubscriberAck()を呼び出しているものを含む、再送信イベントブロックを受信します。
- 保証された配信可能なサブスクライバが肯定応答接続を確立できなかった場合、構成可能な最大再試行回数まで設定可能な速度で再試行します。
- 保証された配信可能パブリッシャが ID を含むイベントブロックを注入し、その ID がパブリッシュクライアントの非確認 ID リストに存在するとします。この場合、インジェクトコールはパブリッシュクライアントによって拒否されます。パブリッシュクライアントが新しいパブリッシャの ACK/NACK コールバック関数に ID を渡すと、ID はリストから消去されます。

保証配信成功のシナリオ

保証された配信のコンテキストでは、パブリッシャとサブスクライバは、データフローのエンドポイントである顧客アプリケーションです。サブスクライブおよびパブリッシュクライアントは、パブリッシャおよびサブスクライバによってパブリッシュ/サブスクライブ API を実装するイベントストリーム処理コードです。

保証配信成功のシナリオの流れは次のとおりです。

1. パブリッシャはパブリッシュ・クライアントにイベント・ブロックを渡します。パブリッシュ・クライアントでは、イベント・ブロックの ID フィールドがパブリッシャによって設定されています。パブリッシュクライアントは、ホストポートフィールドに入力し、未確認の ID リストに ID を追加し、それをエンジンに注入します。
2. イベントブロックはエンジンによって処理され、サブスクライブウィンドウの挿入、更新、または削除はすべてのサブスクライブクライアントに 転送されます。
3. 保証された配信対応サブスクライブクライアントは、イベントブロックを受信し、標準サブスクライバコールバックを使用してサブスクライバに渡します。
4. すべての処理が完了すると、サブスクライバは、イベントブロックポインタで新しい API 関数を呼び出して肯定応答をトリガします。
5. サブスクライブするクライアントは、イベントブロック ID を、イベントブロック内のホストまたはポートに一致する保証された配信肯定応答接続で送信し、エンジンを完全にバイパスします。
6. 肯定応答を受信すると、パブリッシュクライアントは、このイベントブロックに対して受信した肯定応答の数を増分します。その番号が起動時にパブリッシュクライアントに渡されたしきい値に達した場合、パブリッシュクライアントはパラメータが確認され、ID で新しい保証された配信コールバックを呼び出します。承認されていない ID のリストから ID を削除します。

保証配信失敗のシナリオ

表 66 保証配信フローの失敗シナリオ

シナリオ	説明
イベントブロックのタイムアウト	■ イベントブロック固有のタイマーは、配信保証付きパブリッシュクライアントで期限切れになり、このイベントブロックで受信した肯定応答の数が必要なしきい値を下回っています。 ■ パブリッシュクライアントは、パラメータ NACK および ID を使用して、新しい保証型配信コールバックを呼び出します。このイベントブロックに対して、パブリッシュクライアントまたはサブスクライブクライアントによる更なる再送信やその他のリカバリの試みは行われません。パブリッシャはこのイベントブロックをバックアウトして再送信する可能性が最も高いです。 ■ パブリッシュクライアントは、承認されていない ID のリストから ID を削除します。
無効な保証された配信確認接続試行	■ 保証配信対応のパブリッシュクライアントは、保証された配信確認済みサーバーで接続試行を受け取りますが、必要なクライアント接続の数は既に満たされています。 ■ パブリッシュクライアントは接続を拒否し、ERROR メッセージを記録します。 ■ 保証された配信有効サブスクライブクライアントによって 受信された後続のイベントブロックに対して、ERROR メッセージが記録されます。
無効なイベントブロック ID	■ 保証配信可能パブリッシャは、パブリッシュクライアントの未承認 ID リストに既に存在する ID を含むイベントブロックを挿入します。 ■ インジェクトコールはパブリッシュクライアントによって 拒否され、ERROR メッセージが記録されます。

配信保証の API メソッドのパブリッシュ/サブスクライブ

パブリッシュ/サブスクライブ API は、保証された配信セッションを実装するための次のメソッドを提供します。

- C_dfESPGDpublisherStart()
- C_dfESPGDsubscriberStart()
- C_dfESPGDsubscriberAck()
- C_dfESPGDpublisherCB_func()

- C_dfESPGDpublisherGetID()

機能の保証された配信バージョンのないパブリッシュ/サブスクライブ操作では、標準のパブリッシュ/サブスクライブ API 関数を呼び出します。

構成ファイルの内容

パブリッシュクライアントおよびサブスクライブクライアントは、起動時に構成ファイルを読み取り、保証された配信のためのクライアント固有の構成情報を取得します。これらのファイルの形式は次のとおりです。

表 67 配信可能なパブリッシャ設定ファイルの内容を保証

保証された配信確認接続サーバーのローカルポート番号。
否定応答を生成するためのタイムアウト値(秒単位)。
タイムアウト期間内に必要な受信肯定応答の数で、否定応答の代わりに肯定応答を生成します。
ファイル形式 GDpub_port=<port> GDpub_timeout=<timeout> GDpub_numSubs=<number of subscribers generating acknowledged>

表 68 保証された配信有効サブスクライバ構成ファイルの内容

配信可能なパブリッシャホストまたはポートエントリの保証リスト。各エントリは、サブスクライバが保証された配信イベントブロックを受信したい保証された配信可能パブリッシャに対応するホスト:ポートのペアを含む。
肯定応答接続の再試行間隔(秒単位)。
肯定応答接続の最大再試行回数。
ファイル形式 GDsub_pub=<host:port> GDsub_retryInt=<interval> GDsub_maxRetries=<max>