SAS® Event Stream Processing 5.1: 概要

製品の概要

SAS Event Stream Processing により、プログラマは、連続して流れる多数のイベントを迅速に処理して分析できるアプリケーションを構築できます。XML や C++でアプリケーションを書くことができます。イベントストリームは、C、JAVA、または Python パブリッシュ/サブスクライブ API、コネクタクラス、アダプタ実行可能ファイル、Streamviewer、または SAS Event Stream Processing Studio を使用するアプリケーションでパブリッシュされます。

新規または既存のアプリケーション内に専用スレッドプールを持つ Event Stream Processing エンジンを組み込むことができます。ESP クライアントを使用して、Event Stream Processing エンジンの定義(プロジェクト)を ESP サーバーに送り込むことができます。

Event Stream Processing アプリケーションは、通常、イベントストリームに対してリアルタイムな分析を実行します。これらのストリームは、Event Stream Processing エンジンに継続的にパブリッシュされます。Event Stream Processing の一般的な使用例には、以下のものが含まれますが、これらに限定されません。

- センサーデータの監視と管理
- 運用システムの監視と管理
- サイバーセキュリティ分析
- 資本市場取引システム
- 不正検知と防止
- パーソナライズドマーケティング

Event Stream Processing を使用すると、待機時間の少ない増分結果が重要な、長期間にわたって連続的に流れるデータを分析できます。

SAS Cloud Analysis Services を使用した SAS Event Stream Processing

CAS（SAS Cloud Analytic Services）の SAS Event Stream Processing は、2 つのアクションセットで構成されています。

アクションセット	説明
loadStreams	Event Stream Processing プロジェクトを照会し、ストリームデータのスナップショットを取得するアクションを提供します。シングルパスアクションは、Event Stream Processing データを直接消費し、最初にデータをテーブルに入れることなく分析します。 このアクションセットは、SAS Viya を使用するすべてのシステムに 自動的にインストールされます。
espCluster	ESP サーバーのクラスターを開始および表示するためのアクションを提供します。クラスタマネージャと一緒に使用すると、データストリームを分割し、グリッド上で実行されている特定の ESP サーバーにブランチを向けることができます。シングルパスアクションは、このデータを消費し、最初にデータを表に入れることなく分析を実行できます。 個別に発注可能なソフトウェアアプリケーションを使用して、このアクションセットを取得します。

詳細は、“SAS Cloud Analytic Services のアクションでの SAS Event Stream Processing の使用” (SAS Event Stream Processing: 高度なトピック)を参照してください。

イベントストリーム処理アプリケーションの設計

概念的には、イベントは、フィールドの集合として記録できる、特定可能な時間に発生するものです。イベントストリーム処理アプリケーションを設計するときは、次の質問に答える必要があります。

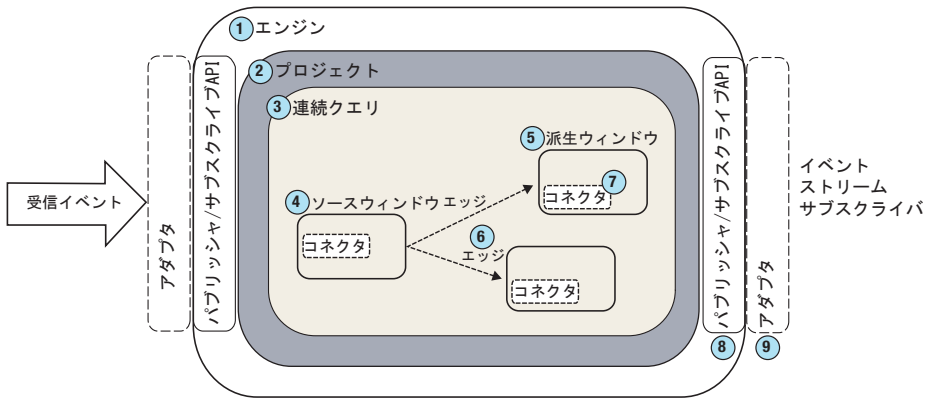
- アプリケーションにはどのようなイベントストリームが、どのようなプロトコルやフォーマットで公開されますか？
- データはどうなりますか？イベントストリームはどのように変換され、分析されますか？
- 結果として得られるイベントストリームは何ですか？どのようなアプリケーションがこれらのイベントストリームをどのフォーマットとプロトコルでサブスクライブしていますか？

これらの質問に対するあなたの答えによって、あなたのモデルの構造が決定します。

イベントストリーム処理モデルとは

Event Stream Processing *モデル*は、パブリッシャからの入力イベントストリームを、サブスクライバが結果的に消費する、意味のあるイベントストリームに変換し分析する方法を指定します。次の図は、モデル階層を示しています。

図 A.1 イベントストリーム処理モデル階層



1. モデル階層の最上位に*エンジン*があります。各モデルには、固有の名前を持つエンジンインスタンスが 1 つだけ含まれています。ESP サーバーはエンジンインスタンスです。
2. エンジンには、それぞれ固有の名前が付けられた 1 つ以上の*プロジェクト*が含まれています。プロジェクトは、サイズがプロジェクト属性として定義されている専用*スレッドプール*で実行されます。スループットのスケーラビリティのために、プロジェクトをネットワークインターフェイスに分散できるように、ポートを指定できます。プロジェクトでスレッドプールを使用すると、Event Stream Processing エンジンは複数のプロセッサコアを使用してより効率的な並列処理を行うことができます。
3. プロジェクトは 1 つまたは複数の*連続クエリ*を含むコンテナです。連続クエリは有向グラフで表されます。このグラフは、1 つまたは複数のパラレルパスの方向に続く結合ノードのセットです。連続クエリは、データ変換であり、受信イベントストリームの分析であるデータフローです。
4. 各クエリーには固有の名前が付けられ、1 つ以上の*ソースウィンドウ*から始まります。
5. ソースウィンドウは、通常、1 つまたは複数の*派生ウィンドウに*接続されます 。派生ウィンドウは、データ内のパターンを検出したり、データを変換したり、データを集計したり、データを分析したり、データに基づいて計算を実行したりすることができます。それらは他の派生ウィンドウに接続することができます。
6. Windows は*エッジ*によって接続されており、 それは関連する方向を持っています。
7. *コネクタ*は、イベントストリームをエンジンとの間でパブリッシュまたはサブスクライブします。コネクタはエンジンに対してインプロセスになっています。
8. *パブリッシュ/サブスクライブ API* を使用して、同じマシンまたはネットワーク上の別のマシンからイベントストリームウィンドウにサブスクライブすることができます。同様に、パブリッシュ/サブスクライブ API を使用して、実行中のイベントストリームプロセッサ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ことができます。
9. *アダプタ*は、ネットワーク接続可能なスタンドアロンの実行可能プログラムです。アダプタは、パブリッシュ/サブスクライブ API を使用して、以下を行うイベントストリームをパブリッシュします。
   - 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します。
   - 任意のウィンドウからイベントストリームをサブスクライブします。

モデリング層のいくつかのオブジェクトは、時間間隔をマイクロ秒単位で測定します。次の間隔はミリ秒単位で測定されます。

- パターンのタイムアウト期間
- 時間ベースの保持における保持期間
- 周期的なウィンドウ出力のためのパルス間隔

ほとんどの非リアルタイムなオペレーティングシステムは、約 10 ミリ秒の割り込み粒度を持っています。したがって、10 ミリ秒より短い時間間隔を指定すると、予測できない結果につながる可能性があります。

注: 実際には、これらの間隔の最小値は 100 ミリ秒でなければなりません。値を大きくするほど予測可能な結果が得られます。

イベントの理解

イベントとは

イベントは、イベントストリームの個々のレコードです。これは、イベントストリーム処理の基本ブロックです。イベントは、メタデータとフィールドデータで構成されます。

イベントのメタデータは、次のもので構成されます。

- オペコード(演算コード)

- (イベントが、保持ポリシー管理からの標準、部分更新、または保持生成イベントかどうかを示す)フラグのセット
- 待機時間測定に使用できる 4 つのマイクロ秒のタイムスタンプのセット

表 A.1 SAS Event Stream Processing でサポートされる演算コード

演算コード	説明
削除(D)	ウィンドウからイベントデータを削除します。
挿入（I）	ウィンドウにイベントデータを追加する
更新（U）	ウィンドウ内のイベントデータを変更する
アップサート（P）	キーフィールドがすでに存在する場合は、イベントデータを更新します。それ以外の場合は、イベントデータをウィンドウに追加します。
安全な削除 (SD)	イベントが存在しない場合、ERROR を生成せずにウィンドウからイベントデータを削除します。

イベントの 1 つまたは複数のフィールドをプライマリキーとして指定する必要があります。キーフィールドは、演算コードのサポートを可能にします。

イベントオブジェクト内のデータは、スキーマオブジェクトに記述された通りの内部形式で格納されます。すべてのキー値は連続しており、イベントの先頭にパックされています。イベントオブジェクトは、作成されたキーに基づいて内部ハッシュ値を保持します。さらに、dfESPeventcomp namespace には、同じ基礎となるスキーマを使用して作成されたイベントを、簡単に比較するための関数があります。

パブリッシュするときに、イベントに更新または挿入の演算コードが必要かどうかわからないときは、アップサートを使用します。イベントがインジェクトされているソースウィンドウは、挿入または更新として処理されるかどうかを決定します。ソースウィンドウは、正しいイベントと演算コードをモデル内の次の接続ウィンドウセットまたはサブスクライバに伝播します。

イベントのデータ型

イベントは、次のデータ型をサポートします。

- INT32
- INT64
- DOUBLE
- UTF8STR（文字列）
- DATETIME（秒単位の粒度）
- TIMESTAMP（マイクロ秒単位までの粒度）
- MONEY（192 ビット固定小数）
- BINARY（バイナリラージオブジェクトまたは BLOB）
- RUTF8STR（参照カウント文字列、または rstring）
- ARRAY（32 ビット整数、64 ビット整数、double）

基本データ型(INT32、INT64、DOUBLE、UTF8STR、DATETIME、TIMESTAMP、および MONEY)のデータは、イベントのインラインで格納されます。インラインストレージは、非常に高速なインデックス作成とシリアル化を可能にします。

BINARY、RUTF8STR、および ARRAY データ型はインラインで格納されません。代わりに、イベントで参照されます。つまり、イベントは実際のデータへのポインタを保持します。これらのデータ型は、イベントのキーフィールドとして使用することはできません。それらはオブジェクトレベルで参照カウントされます。これにより、同じオブジェクトを複数のイベントで参照できるようになり、メモリ使用量が減少し、オブジェクトを新規作成してデータをコピーするのにかかる時間が短縮されます。

キー以外のフィールドの場合、RUTF8STR データ型は UTF8STR(文字列)よりも経済的です。RUTF8STR はすべてのウィンドウの内部と外部に渡すことができます。使用するたびに内部的に標準文字列として参照されます。これらの特性は、メモリの大幅な節約につながります。UTF8STR データ型は、イベントのインラインで格納されることに注意してください。16K 文字列が一連のウィンドウを介して伝播されると、ストリングのコピーが各ウィンドウ内のイベントに含まれます。代わりに RUTF8STR を使用すると、最初に文字列を保持する dfESPrstring オブジェクトが作成されます。各イベントには、そのオブジェクトへの 8 バイトのポインタが含まれます。

詳細は、“dfESPdatavar” (SAS Event Stream Processing: プログラミングリファレンス)を参照してください。

イベントのバイナリコードへの変換

イベントがソースウィンドウに公開されると、高速フィールドポインタと制御情報を持つバイナリコードに変換されます。この変換により、スループットのパフォーマンスが向上します。

CSV イベントのファイルまたはストリームをファイルまたはバイナリイベントのストリームに変換できます。このファイルまたはストリームをプロジェクトに公開して、CSV ファイルまたはストリームよりも高速で処理することができます。

バイナリへの CSV 変換は CPU を大量に使用するため、ファイルを 1 度変換するか、ソースでストリームを変換することをお勧めします。実際のプロダクションアプリケーションでは、頻繁に受信されるデータは何らかの種類のバイナリ形式であり、SAS Event Stream Processing で使用するために必要なのは再切り取りのみです。それ以外の場合、データはバイナリに変換する必要があるテキストとして提供されます。

イベントの文字列フィールドを正しく表現するには、対応する CSV 文字列フィールドが次の規則に従わなければなりません。

- 文字列フィールドに先頭または末尾の空白が含まれる場合は、文字列フィールド全体を二重引用符で囲む必要があります。
- 文字列フィールドに CSV 区切り文字(デフォルトで'、')が含まれている場合は、文字列フィールド全体を二重引用符で囲む必要があります。
- 先頭のエスケープ文字('\')を含む文字列フィールドには、リテラルの二重引用符(")文字を前に置く必要があります。
- 先頭のエスケープ文字('\')を含む文字列フィールドには、リテラルエスケープ('\')文字を前に置く必要があります。
- 複数バイトの"Byte-Order Mark"(BOM)シーケンスはサポートされていません。これを含めると、CSV 文字列の適切な解析が妨げられます。
- 改行('\n')のシーケンスはサポートされていません。CSV は行指向の形式であり、改行は行のデリミタとして予約されています。

バイナリへの CSV 変換については、SAS Event Stream Processing インストールの **examples/cxx** ディレクトリにあるサンプルアプリケーション"csv2bin"を参照してください。**$DFESP_HOME/examples** の readme.examples.txt ファイルは、この例を使用して CSV ファイルをイベントストリームプロセッサバイナリファイルに変換する方法を説明しています。この例では、C クライアント API のメソッドを使用して C++で変換を実行する方法を示します。また、Java または Python クライアント API を使用して変換することもできます。

次のコード例は、stdin からバイナリイベントを読み込み、実行中のプロジェクトにイベントをインジェクトします。

注: 特定のファイルに存在できるのは、1 つのウィンドウのイベントのみです。たとえば、すべてのイベントは同じソースウィンドウ用でなければなりません。また、64 個の入力イベントのブロックにデータをグループ化して、余分な待機時間を生み出すことなくオーバーヘッドを削減します。

```
// For windows it is essential that you read binary
// data in BINARY mode.
//
dfESPfileUtils::setBinaryMode(stdin);
  // Trade event blocks are in binary form and
  // are coming using stdin.
    while (true)
{
  // more trades exist
  // Create event block.
    ib = dfESPeventblock::newEventBlock(stdin,
        trades->getSchema());
     if (feof(stdin))
        break;
     sub_project->injectData(subscribeServer,
                        trades, ib);
}
sub_project->quiesce(); // Wait for all input events to be processed.
```

イベントブロックについて

イベントブロックには 0 個以上のバイナリイベントが含まれ、パブリッシュ/サブスクライブクライアントはエンジンとの間でイベントブロックを送受信します。パブリッシュ/サブスクライブ操作ではオーバーヘッドが発生するため、複数のイベント(イベントブロックあたり 512 イベントなど)を含むイベントブロックを処理すると、待機時間に対する影響を最小限に抑えてスループットのパフォーマンスが向上します。

イベントブロックは、トランザクションまたは標準にすることができます。

イベントブロック	説明
トランザクション	プロジェクトを通じた処理はアトミックです。イベントブロック内の 1 つのイベントが失敗した場合(たとえば、存在しないイベントの削除など)、イベントブロック内のすべてのイベントが失敗します。失敗したイベントは記録され、オプションの不良レコードファイルに格納されます。これはさらに処理を行うことができます。
標準	プロジェクトを通じた処理はアトミックではありません。イベントは効率のためにパッケージされていますが、ソースウィンドウにインジェクトされると個別に処理されます。

ユニークなトランザクション ID は、変換されたイベントブロックがエンジンモデルを通過する際に伝播されます。この持続性により、イベントストリームサブスクライバは、サブスクリプションされたイベントのグループを、ID を介して特定のパブリッシュされたイベントのグループに戻すことができます。

エンジンの実装

エンジンはイベントストリーム処理モデルの最上位のコンテナです。各モデルには、固有の名前を持つエンジンインスタンスが 1 つだけ含まれています。エンジンは、スタンドアロンの実行可能ファイルとしてインスタンス化することも、C++モデリングレイヤーを使用してアプリケーション内に組み込むこともできます。

SAS Event Stream Processing は、エンジンを実装するための 3 つのモデリング API を提供します。

- SAS Event Stream Processing Studio を使用して、イベントストリーム処理モデルを作成し、それらのモデルに基づいて XML コードを生成することができます。

- XML レイヤーを使用すると、単一のエンジン定義を定義したり、動的なプロジェクトの作成と削除を行うエンジンを定義することができます。SAS Visual Scenario Designer などの他の製品で XML レイヤーを使用して、ビジュアルモデリングを実行できます。
- C++Modeling API を使用すると、アプリケーションプロセススペース 内に Event Stream Processing エンジンを組み込むことができます。また、アプリケーションのメインスレッドがエンジンと直接対話できるようにする低レベルの関数も提供します。

XML レイヤーでは、ESP サーバーは、プロジェクトとエンジンの Event Stream Processing の定義を受け入れるエンジンプロセスです。ESP サーバーはインスタンス化されたエンジンであるため、送信するエンジン仕様はすべて無視されます。代わりに、提出されたエンジン内でプロジェクトをインスタンス化します。

アプリケーションプロセス内に Event Stream Processing エンジンを C++Modeling API を介して組み込むことができます。エンジンを含むアプリケーションプロセスは、サーバーシェルでも、エンジンスレッドと対話する実際のアプリケーションスレッドでもかまいません。

複数のプロジェクトを実装するか、複数の連続クエリを実装するかは、処理のニーズに応じて決まります。ESP サーバーでは、レイヤーがサービスとして使用されているため、複数のプロジェクトを動的に導入、破棄、停止、または開始することができます。すべてのモデリングレイヤーでは 、異なるユースケースや単一のエンジンインスタンスで異なるスレッドモデルを取得するために、複数のプロジェクトを使用できます。以下を使用することができます。

- より高いレベルの決定性のための単一スレッドモデル
- より高いレベルの並列性のためのマルチスレッドモデル

モジュール式のメカニズムとして連続クエリを使用できるため、実装するクエリの数は、ウィンドウがどのように区画化されているかによって 異なります。連続クエリでは、必要な数のウィンドウをインスタンス化して定義できます。任意のウィンドウが 1 つまたは複数のウィンドウにデータを流すことができます。連続クエリでは、ループバック条件は許可されません。プロジェクトコネクタまたはアダプタを使用して、連続クエリ全体をループバックすることができます。

イベントストリームは、次のいずれかを使用してソースウィンドウにパブリッシュまたはインジェクトする必要があります。

- パブリッシュ/サブスクライブ API
- コネクタ
- アダプタ
- HTTP クライアント
- SAS Event Stream Processing Studio
- Streamviewer
- C++モデリング API の continuous-query-inject メソッド

連続クエリでは、使用可能なすべてのウィンドウタイプを 使用してデータフローモデルを定義できます。プロシジャウィンドウを使用すると、C++または DS2 を使用してイベントストリーム入力ハンドラーを書き込むことができます。

DS2 で作成された入力ハンドラは SAS Threaded Kernel ライブラリの機能を使用できるため、既存の SAS モデルをプロシジャウィンドウで実行できます。既存のモデルが本質的に追加方式で、1 度に 1 つのイベントを処理できる場合にのみ、これを実行できます。

プロジェクトの理解

*プロジェクト*は、1 つ以上の連続クエリを保持するコンテナを指定し、ユーザ定義のサイズのスレッドプールによってサポートされます。プロジェクトは、増分計算のための決定性のレベルを指定することができます。パブリッシュ/サブスクライブのスケーラビリティのためのオプションのポートを指定することもできます。

データフローモデルは常に計算的に決定性です。プロジェクトがマルチスレッド化されている場合、異なるプロジェクト実行間で異なる時間に中間計算が行われる可能性があります。したがって、プロジェクトがすべての増分計算を監視する場合、インクリメンタル計算の統一が常に同じであっても、増分は実行によって変化する可能性があります。

注: 指定された決定性レベルまたはエンジンで使用されるスレッドの数にかかわらず、各ウィンドウは常にすべてのデータを順番に処理します。したがって、ウィンドウによって受信されたデータは、決して並べ替えられず、順番以外では処理されません。

連続クエリについて

連続クエリは、1 つ以上の有向グラフのウィンドウを保持し、ウィンドウ間の接続を指定できるコンテナを指定します。連続クエリ内のウィンドウは、データを変換または分析したり、パターンを検出したり、計算を実行することができます。クエリコンテナは、大規模プロジェクトに対して機能的なモジュール性を提供します。通常、各コンテナは単一の有向グラフを保持します。

連続クエリの処理は、次の手順に従います。

1 1 つまたは複数のイベントを含むイベントブロック(アトミックプロパティあり/なし)がソースウィンドウにインジェクトされます。

2 イベントブロックは、ソースウィンドウに直接結合されている任意の派生ウィンドウに流れます。トランザクションプロパティが設定されている場合、1 つ以上のイベントのイベントブロックは、結合された各派生ウィンドウへの途中でアトミックに処理されます。つまり、すべてのイベントを完全に実行する必要があります。

   トランザクションプロパティを持つイベントブロック内のイベントが失敗すると、そのイベントブロック内のすべてのイベントが失敗します。失敗したイベントが記録されます。この機能を有効にすると、レビュー、修正、および再発行するために、不正レコードファイルに書き込まれます。

3 派生ウィンドウは、イベントを派生ウィンドウのプロパティに基づいてゼロまたは複数の新規作成イベントに変換します。新規作成イベントが派生ウィンドウによって計算された後、イベントはモデルの更に先に流れ、新規作成イベントが潜在的に計算される場所である、接続された派生ウィンドウの次のレベルに到達します。

4 このプロセスは、次のいずれかが発生したときに、特定のイベントブロックのモデルのアクティブパスごとに終了します。

   - 生成されたイベントを渡すことができる、接続された派生ウィンドウはこれ以上存在しません。
   - パスに沿った派生ウィンドウが、そのイベントブロックに対して結果として発生したイベントはゼロです。したがって、結合された派生ウィンドウの次のセットには何も渡されません。

ウィンドウについて

連続クエリには、*ソースウィンドウ*と 1 つ以上の*派生ウィンドウ*が含まれます 。Windows は*エッジ*によって接続されており、 それは関連する方向を持っています。

SAS Event Stream Processing は、次のウィンドウタイプをサポートしています。

ウィンドウタイプ	説明
ソースウィンドウ	すべてのイベントストリームは、パブリッシュされたり、ソースウィンドウにインジェクトされたりして、連続クエリを入力する必要があります。イベントストリームを他のウィンドウタイプにパブリッシュまたはインジェクトすることはできません。
計算ウィンドウ	入力イベントストリームフィールドの計算操作を通じて、入力イベントを出力イベントに 1 対 1 に変換できるようにします。
コピーウィンドウ	親ウィンドウのコピーを作成します。コピーを作成すると、新規作成イベント状態保持ポリシーを設定することができます。保持ポリシーは、ソースウィンドウおよびコピーウィンドウでのみ設定できます。 コピーウィンドウのイベント状態の保持は、ウィンドウが挿入専用ではなく、ウィンドウインデックスが pi_EMPTY に設定されていない場合にのみ設定できます。すべての子ウィンドウは保持管理に影響されます。イベントは、Windows 保持ポリシーを超えると削除されます。
集計ウィンドウ	非キーフィールドが計算されるという点で、計算ウィンドウと同様です。集計ウィンドウは、グループごとの条件のキーフィールドを使用します。すべてのユニークキーフィールドの 組み合わせは、集計ウィンドウ内で独自のグループを形成します。同じキーの組み合わせを持つすべてのイベントは、同じグループの一部です。
カウンタウィンドウ	モデルを通ってストリーミングされているイベントの数とそれらが処理されているレートを確認できます。
フィルタウィンドウ	登録されたブールフィルタ関数または式を使用します。この関数または式は、フィルタウィンドウにどの入力イベントが許可されるかを決定します。
関数ウィンドウ	さまざまな種類の関数を使用して、イベント内のデータを操作または変換できます。機能ウィンドウ内のフィールドは階層構造にすることができ、ウェブ分析などの応用に役立ちます。
ジオフェンスウィンドウ	ウィンドウを作成して、イベントストリームの場所が関心領域の内側にあるのか、関心領域に近いのかを判断できます。
結合ウィンドウ	2 つの入力ウィンドウと結合タイプをとります。1 対多、多対 1、多対多の等価結合をサポートします。内部結合と外部結合の両方がサポートされています。
通知ウィンドウ	メール、ショートメッセージまたはマルチメディアメッセージを 使用して通知が送信できます。任意の数の配布チャネルを作成して通知を送信することができます。通知ウィンドウは、機能ウィンドウと同じ基本言語と関数を使用します。
パターンウィンドウ	目的のイベント(EOI)の検出を有効にします。このウィンドウタイプで定義されたパターンは、宣言された関心のあるイベントを論理的に接続する式です。 パターンウィンドウを定義するには、関心事のイベントを定義し、次にこれらの関心のあるイベントを演算子を使用して接続する必要があります。サポートされる演算子は、"AND"、"OR"、"FBY"、"NOT"、"NOTOCCUR"、および"IS"です。演算子はオプションの一時的条件を受け入れることができます。
プロシジャウィンドウ	入力ウィンドウの任意の数や各入力ウィンドウの入力ハンドラの仕様設定を可能にします。
テキストカテゴリウィンドウ	入力イベントのテキストフィールドを分類できます。テキストカテゴリウィンドウは挿入専用です。テキストフィールドは、スコアを持つ 0 個以上のカテゴリを生成することができます。 このオブジェクトにより、SAS Contextual Analysis のライセンスを取得したユーザーは、MCO ファイルを使用してテキストカテゴリウィンドウを初期化できます。

ウィンドウタイプ	説明
テキストコンテキストウィンドウ	非構造化文字列フィールドから分類された用語の抽象化を有効にします。 このオブジェクトを使用すると、SAS Contextual Analysis のライセンスを取得したユーザーは Liti ファイルを使用してテキストコンテキストウィンドウを初期化できます。このウィンドウタイプを使用して、イベントの入力からの文字列フィールドを分析し、分類された用語を検索します。これらの用語から生成されたイベントは、他のウィンドウタイプで分析することができます。例えば、パターンウィンドウは、テキストコンテキストウィンドウの後に、関心のあるツイートパターンを探すことができます。
テキストセンチメントウィンドウ	指定された入力テキストフィールド内のテキストの感情とその出現の確率を決定します。センチメント値は"正"、"中立"または"負"です。確率は 0 と 1 の間の値です。テキストセンチメントウィンドウは挿入専用です。 このオブジェクトを使用すると、SAS Sentiment Analysis のライセンスを取得したユーザーは、SAM Sentiment Analysis を使用してテキストセンチメントウィンドウを初期化できます。
テキストトピックウィンドウ	イベントで SAS Text Miner 分析を実行します。テキストトピックウィンドウは、ドキュメントから文字列フィールドとしてテキストを受け取り、処理します。テキストマイニング分析モデルは、分析ストア(ASTORE)ファイルを使用してテキストトピックウィンドウに入ります。
結合ウィンドウ	1 つ以上のストリームを同じスキーマとマージする簡単な結合を指定します。

SAS Event Stream Processing Analytics は、追加の一連のウィンドウタイプを提供します。詳細は、*SAS Event Stream Processing: SAS Event Stream Processing Analytics の使用*を参照してください。

ソフトウェアライセンスの更新

SAS Event Stream Processing を使用するアプリケーションを実行するには、有効なライセンスファイルが必要です。

ソフトウェアライセンス更新確認メッセージにライセンスファイルが含まれています。そのファイルを **$DFESP_HOME/etc/license** にコピーします。

注: SAS Event Stream Processing 5.1 の場合、UNIX の場所は**$DFESP_HOME/etc/license** です。**$DFESP_HOME** は**/opt/sas/viya/home/SASEventStreamProcessingEngine/5.1.0** です。ウィンドウの場所は**%DFESP_HOME%\ etc \ license** です。ここでは、**%DFESP_HOME%**は Windows 上のインストールディレクトリです。

アプリケーションが C++モデリング API dfESPlibrary::Initialize を使用してライセンスの場所を変更した場合、その場所にライセンスファイルをコピーします。

イベントストリーム処理アプリケーションの作成

1 エンジンを作成し、ESP サーバーまたは C++アプリケーション内でインスタンス化します。

ESP サーバーの詳細については、“ESP サーバーの使用” (SAS Event Stream Processing: ESP サーバーの使用)を参照してください。

アプリケーションの記述に使用する C++キーモデリングオブジェクトの詳細については、“SAS Event Stream Processing のための C++モデリングオブジェクト” (SAS Event Stream Processing: プログラミングリファレンス)を参照してください。

2 パブリッシュ/サブスクライブ API、コネクタ、アダプタ、SAS Event Stream Processing Studio、Streamviewer を使用するか、C++モデル用の dfESPcontquery::injectEventBlock() メソッドを使用して、1 つ以上のイベントストリームをエンジンにパブリッシュします。

次のいずれかの方法でパブリッシュおよびサブスクライブできます。

- Java、C、または Python パブリッシュ/サブスクライブ API を使用します。
- パブリッシュ/サブスクライブ API、SAS Event Stream Processing Studio、または Streamviewer を使用するパッケージ化されたコネクタ(インプロセスクラス)またはアダプタ（ネットワーク化された実行可能ファイル）を使用します。
- サブスクライブまたは連続クエリのインジェクションイベントブロック方式のインプロセスコールバックを使用します。

パブリッシュ/サブスクライブ API の詳細については、“パブリッシュ/サブスクライブ API の概要” (SAS Event Stream Processing: パブリッシュ/サブスクライブ API)を参照してください。

コネクターは、イベントストリームプロセッサと同じプロセススペースでインスタンス化される C++クラスです。コネクタは、XML モデルまたは C++モデル内から使用できます。アダプタは、対応するコネクタクラスを使用して、パブリッシュ/サブスクライブ API を使用するスタンドアロンの実行可能ファイルを提供します。したがって、ネットワーク化することができます。詳細は、“コネクタとアダプタの使用” (SAS Event Stream Processing: コネクタとアダプタ)を参照してください。

3 パブリッシュ/サブスクライブ API、コネクタ、アダプタ、SAS Event Stream Processing Studio、Streamviewer を使用するか、C++モデルの dfESPwindow::addSubscriberCallback()メソッドを使用して、連続クエリ内の関連する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。

-b*filename* を使用して C++Event Stream Processing アプリケーションを開始すると、アプリケーションは、計算上の失敗のために処理されなかったイベントを、指定されたログファイルに書き込みます。このオプションを指定しないと、同じデータが stderr に書き込まれます。不正なイベントのログを作成して、新規挿入を監視できるようにすることをお勧めします。