SYBASE®
An SAP® Company

アダプタ・ガイド

Sybase Event Stream Processor 5.0

ドキュメント ID： DC01736-01-0500-01
改訂： 2011 年 12 月

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

目次

第 1 章　はじめに

Sybase® Event Stream Processor では、データをサブスクライブ、パブリッシュするために使用できる入力アダプタと出力アダプタが多く提供されており、さらに外部プロセス・アダプタも構成できます。 また、Event Stream Processor にはアダプタを記述できる SDK も複数用意されています。

同梱されているアダプタは、多数のデータ型をサポートし、サポートされていないデータ型マッピングも提供します。

入力アダプタと出力アダプタ

入力アダプタと出力アダプタによって、Event Stream Processor は動的なまたは静的な外部ソースと送信先との間でメッセージを送受信できます。

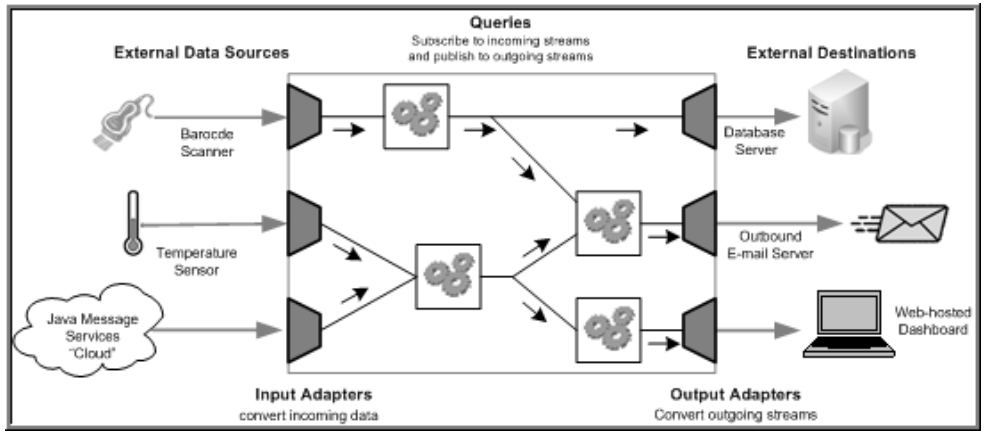
外部のソースまたは送信先には、以下があります。

- データ・フィード
- センサ・デバイス
- メッセージング・システム
- RFID (Radio frequency identification) リーダ
- 電子メール・サーバ
- リレーショナル・データベース

入力アダプタは外部データソースに接続し、外部ソースからの受信メッセージを、Event Stream Processor サーバによって受け入れられるフォーマットに変換します。出力アダプタは、Event Stream Processor によってパブリッシュされたローを外部の送信先と互換性のあるメッセージ・フォーマットに変換し、それらのメッセージを下流のストリームに送信します。

以下の図は、温度センサ、バー・コード・スキャナ、JMS (Java Message Service) クラウドからのメッセージを、Event Stream Processor と互換性のあるフォーマットに変換する、一連の入力アダプタを示しています。 データが Event Stream Processor 内のさまざまなクエリを使用して処理されると、生成されたローを出力アダプタが外部のデータベース・サーバ、電子メール・サーバ、Web サービス・ダッシュボードに送信される更新に変換します。

図 1 : Event Stream Processor 内のアダプタ



内部アダプタと外部アダプタ

サーバの一部として実行されるアダプタは、内部アダプタと呼ばれます。 独立したプロセスとして実行されるアダプタは、外部アダプタと呼ばれます。

サーバが少ないオーバヘッドでデータをアダプタから取得 (またはアダプタへ送信) できるため、通常内部アダプタはより速く実行されます。 内部アダプタは、サーバが対応するプロジェクト (クエリ・モジュール) を起動するときに、サーバによって起動されます。 アダプタは、Sybase Event Stream Processor スタジオによって認識されます。アダプタのメニューからアダプタを選択することによって、スタジオ内部からアダプタをストリームにアタッチできます。 内部アダプタのデメリットとして、アダプタがクラッシュした場合、サーバもクラッシュさせてしまう可能性があります。

外部アダプタは、内部アダプタよりも柔軟性があり、サーバ以外のマシンで実行できます。 内部アダプタのように、外部アダプタはスタジオを使用して設定できます。また、アダプタ設定ファイルと共にインストールされている場合にかぎり、サーバから起動、停止できます。

外部アダプタは、「管理型」または「非管理型」のいずれかです。 管理型外部アダプタでは、内部アダプタとほとんど同じように、スタジオを使用した設定、CCL **ATTACH ADAPTER** 文での参照、サーバによる起動と停止が可能なアダプタ設定ファイル (`.cnxml`) を使用できます。 非管理型外部アダプタは、CCL **ATTACH ADAPTER** 文で参照されず、サーバによって管理されません。 これらのアダプタは、個別に起動、停止、設定します。

カスタム・アダプタ

Event Stream Processor で提供されるアダプタ以外に、独自のアダプタを作成してサーバに統合できます。 標準のアダプタが管理できないさまざまな外部要件を処理するように、アダプタを設計できます。

Event Stream Processor ではさまざまな SDK が用意されており、多くのプログラム言語でアダプタを作成できます。以下にサポートされる言語の一部を示します。

- C
- C++
- Java
- .NET (C#、Visual Basic など)

SDK でサポートされているバージョンについては、『インストール・ガイド』を参照してください。

参照：
- *カスタム外部アダプタ* (562 ページ)
- *カスタム内部アダプタ* (549 ページ)
- *Java 外部アダプタ* (572 ページ)
- *C/C++ 外部アダプタ* (576 ページ)
- *.Net 外部アダプタ* (579 ページ)
- *カスタム外部アダプタの作成* (571 ページ)

アダプタの使用

Event Stream Processor によって提供されるアダプタとカスタム・アダプタを使用するときに考慮する事前計画と設定の手順。

入力アダプタをサーバにアタッチする前に実行する標準作業と CCL **ATTACH ADAPTER** 文の概要。

注意： 非管理型アダプタは、CCL **ATTACH ADAPTER** 文では参照されず、個別に設定、起動、停止する必要があります。

個々のアダプタの設定、データ型マッピング、スキーマ検出に関する詳細情報は、このマニュアルに記載されています。『CCL プログラマーズ・ガイド』の CCL クエリ、**ATTACH ADAPTER** 文、**CREATE SCHEMA** 文、**Parameters** に関するトピックも参照してください。

1. 入力データにアクセスします。「取得してサーバに入れるデータのセットまたはサブセットを決定します。
2. タスクで使用する入力アダプタを選択します。
   データソースがサーバによってサポートされないデータ型を使用する場合、サーバはデータを受け入れられるデータ型にマップします。 このマニュアルにあるアダプタの関連マッピングの説明を参照してください。
3. 必要に応じてアダプタを設定します。
4. ストリームまたはウィンドウを作成し、**CREATE SCHEMA** 文を使用して受信データの構造を定義します。
5. **ATTACH ADAPTER** 文を使用してアダプタをサーバ・ストリームまたはウィンドウにアタッチし、アダプタ・プロパティの値を設定します。
   アダプタのプロパティのデフォルト・パラメータを宣言するには、**DECLARE** ブロックと **parameters** 修飾子を使用してデフォルトのパラメータ値を定義してから、アダプタにアタッチします。 **ATTACH ADAPTER** 文を作成すると、アダプタのプロパティを、宣言したパラメータ値に設定できます。

   注意： モジュールまたはオブジェクトがロードされたとき、宣言されたパラメータのみを新しい値にバインドできます。
6. データ・クエリを実行し、分析を実行します。
7. 結果をパブリッシュします。

出力アダプタを使用したデータのパブリッシュ

出力アダプタを外部データの送信先にアタッチする前に実行する標準作業の概要。

1. 出力データにアクセスします。 外部データの送信先に送信するデータのセットまたはサブセットを決定します。
2. タスクで使用する出力アダプタを選択します。
   出力先がサーバによってサポートされないデータ型を使用する場合、サーバはデータを受け入れられるデータ型にマップします。 このマニュアルにあるアダプタの関連マッピングの説明を参照し、結果データ型が外部データの送信先によって許容されることを確認してください。
3. 必要に応じてアダプタを設定します。
4. 出力ストリームまたはウィンドウを作成し、**CREATE SCHEMA** 文を使用して送信データの構造を定義します。
5. **ATTACH ADAPTER** 文を使用してアダプタを出力ストリームまたはウィンドウにアタッチし、アダプタ・プロパティの値を設定します。
   アダプタのプロパティのデフォルト・パラメータを宣言するには、**DECLARE** ブロックと **parameters** 修飾子を使用してデフォルトのパラメータ値を定義して

から、アダプタにアタッチします。 **ATTACH ADAPTER** 文を作成すると、アダプタのプロパティを、宣言したパラメータ値に設定できます。

注意： モジュールまたはオブジェクトがロードされたとき、宣言されたパラメータのみを新しい値にバインドできます。

6. 結果をパブリッシュします。

データ型

Sybase Event Stream Processor では、すべてのコンポーネントで integer、float、string、money、long、timestamp のデータ型がサポートされます。

データ型	説明
`integer`	符号付き 32 ビット整数。 有効な値の範囲は、-2147483648 ～ +2147483647 (-2^{31} ～ 2^{31-1}) です。 この範囲外の定数値は、自動的に long データ型として処理されます。 変数、パラメータ、またはカラムを -2147483648 の値で初期化するには、(-2147483647) -1 を指定して、CCL コンパイラのエラーを回避します。
`long`	符号付き 64 ビット整数。 有効な値の範囲は、-9223372036854775808 ～ +9223372036854775807 (-2^{63} ～ 2^{63-1}) です。 変数、パラメータ、またはカラムを -9223372036854775808 の値で初期化するには、(-9223372036854775807) -1 を指定して、CCL コンパイラのエラーを回避します。
`float`	倍精度 64 ビット数値浮動小数点。 有効な値の範囲は、およそ -10^{308} ～ $+10^{308}$ です。
`string`	UTF-8 でエンコードされたバイト値による可変長文字列。最大文字列長は、プラットフォームによって異なりますが、65535 バイトを超えることはできません。
`money`	国際基準の符号付き 64 ビット整数。 入力データ・ストリームでは、通貨記号とカンマはサポートされません。

データ型	説明
`money(n)`	可変の位取りをサポートする符号付き 64 ビット整数 (小数点以下 1 ～ 15 桁)。 入力データ・ストリームでは、通貨記号とカンマは、サポートされませんが、小数点はサポートされます。 サポートされる値の範囲は、特定の位取りによって変化します。 `money(1)`：-922337203685477580.8 ～ 922337203685477580.7 `money(2)`：-92233720368547758.08 ～ 92233720368547758.07 `money(3)`：-9223372036854775.808 ～ 9223372036854775.807 `money(4)`：-922337203685477.5808 ～ 922337203685477.5807 `money(5)`：-92233720368547.75808 ～ 92233720368547.75807 `money(6)`：-92233720368547.75808 ～ 92233720368547.75807 `money(7)`：-922337203685.4775808 ～ 922337203685.4775807 `money(8)`：-92233720368.54775808 ～ 92233720368.54775807 `money(9)`：-9223372036.854775808 ～ 9223372036.854775807 `money(10)`：-922337203.6854775808 ～ 922337203.6854775807 `money(11)`：-92233720.36854775808 ～ 92233720.36854775807 `money(12)`：-9223372.036854775808 ～ 9223,372.036854775807 `money(13)`：-922337.2036854775808 ～ 922337.2036854775807 `money(14)`：-92233.72036854775808 ～ 92233.72036854775807 `money(15)`：-9223.372036854775808 ～ 9223.372036854775807 変数、パラメータ、またはカラムを -92233.72036854775807 の値で初期化するには、(-9...7) -1 を指定して、CCL コンパイラのエラーを回避します。 money 定数の位取りを明示的に指定するには、**Dn** 構文を使用します (**n** は精度を表します)。たとえば、100.1234567D7、100.12345D5 というように指定します。 `money(n)` タイプ間の暗黙的な変換は、範囲または位取りが失われる危険性があるため、サポートされていません。 **cast** 関数を実行して、異なる位取りを持つ money 型を処理します。

データ型	説明
bigdatetime	マイクロ秒精度のタイムスタンプ。 デフォルト・フォーマットは、YYYY-MM-DDTHH:MM:SS:SSSSSS です。 すべての数値データ型は、bigdatetime に暗黙的にキャストされます。 変換規則は、以下のようにデータ型ごとに異なります。 • boolean、integer、long の値はすべて、元のフォーマットで bigdatetime に変換されます。 • money(n) と float の整数部分の値のみ bigdatetime に変換されます。 cast 関数を使用して money(n) と float の値を、精度を指定した bigdatetime に変換します。 • date の値はすべて 1000000 で乗算されてマイクロ秒単位に変換され、bigdatetime フォーマットを満たします。 • timestamp の値はすべて 1000 で乗算されてマイクロ秒単位に変換され、bigdatetime フォーマットを満たします。
timestamp	ミリ秒精度のタイムスタンプ。 デフォルト・フォーマットは、YYYY-MM-DDTHH:MM:SSS です。
date	ミリ秒精度の日付。 デフォルト・フォーマットは、YYYY-MM-DDTHH:MM:SSS です。

データ型	説明
`interval`	2 つのタイムスタンプ間のマイクロ秒数を表す符号付き 64 ビット整数。 スペースで区切られたフォーマットで複数の単位を使用し、`interval` を指定します。たとえば、"5 Days 3 hours 15 Minutes" (5 日と 3 時間 15 分)。 interval カラムに送信される外部データは、マイクロ秒単位であると見なされます。 `interval` 値と `string` データとの間の変換では、単位指定はサポートされていません。 `interval` を指定する場合、適切なマイクロ秒数に変換されるときに、指定した interval が 64 ビット整数 (`long`) に適合している必要があります。 それぞれの `interval` 単位で、マイクロ秒に変換するときに long 型に適合する最大許容値は、以下のとおりです。 • MICROSECONDS (MICROSECOND、MICROS)： +/- 9223372036854775807 • MILLISECONDS (MILLISECOND、MILLIS)： +/- 9223372036854775 • SECONDS (SECOND、SEC)： +/- 9223372036854 • MINUTES (MINUTE、MIN)： +/- 153722867280 • HOURS (HOUR,HR)： +/- 2562047788 • DAYS (DAY)： +/- 106751991 カッコで囲まれた値は、`interval` 単位の代替名です。 単位の最大値が指定されると、その他の単位は指定できません。指定すると、オーバフローが発生します。 それぞれの単位は 1 回だけ指定できます。
`binary`	ロー・バイナリ・バッファを表す。値の最大長は、プラットフォームによって異なりますが、65535 バイトを超えることはできません。 NULL 文字を使用できます。
`boolean`	値は true または false。 `boolean` の許容範囲外の値のフォーマットは、0/1/false/true/y/n/on/off/yes/no で、大文字と小文字は区別されません。

注意： サポートされないデータ型は、サポートされるデータ型にマップされますが、それぞれのデータ型マッピングは、アダプタ・タイプごとに異なります。 使用しているアダプタのデータ型のマッピングについては、『アダプタ・ガイド』の「データ型のマッピング」を参照してください。

アダプタ・パラメータのデータ型

Event Stream Processor によって提供されるアダプタ、または、お客様が作成したカスタム内部アダプタまたはカスタム外部アダプタで使用できるデータ型の包括的なリスト。

カスタム外部アダプタのいくつかの例外が、データ型の説明に記載されています。

注意： この表には、Event Stream Processor によってサポートされているアダプタ関連のデータ型がすべて掲載されています。 アダプタによってサポートされている特定のデータ型と、そのデータ型マッピングの詳細については、アダプタの該当するセクションを参照してください。

データ型	説明
`boolean`	値は、true または false。 `boolean` の有効な範囲以外の値のフォーマットは、0/1/false/true/y/n/on/off/yes/no で、大文字小文字は区別されません。
`choice`	ユーザが 1 つの値を選択するカスタム値のリスト。
`configFilename`	UTF-8 でエンコードされたバイト値を使用する、可変長の文字列。最大文字列長はプラットフォームによって異なりますが、65535 バイトを超えることはできません。
`directory`	UTF-8 でエンコードされたバイト値を使用する、可変長の文字列。最大文字列長はプラットフォームによって異なりますが、65535 バイトを超えることはできません。
`double`	浮動小数点数値。 有効な値の範囲は、2.22507e-308 ～ 1.79769e+308 です。
`filename`	UTF-8 でエンコードされたバイト値を使用する、可変長の文字列。最大文字列長はプラットフォームによって異なりますが、65535 バイトを超えることはできません。
`int`	符号付き 32 ビット整数値。 有効な値の範囲は、-2147483648 ～ +2147483647 (-23^1 ～ 2^31-1) です。 この範囲外の定数値は、自動的に長整数データ型として処理されます。 変数、パラメータ、またはカラムを最小の負値で初期化するには、CCL コンパイラ・エラーを回避するために、代わりに (-2...7) -1 を使用します。 たとえば、変数、パラメータ、またはカラムを -2147483648 で初期化するには、(-2147483647) -1 を指定します。

データ型	説明
password	UTF-8 でエンコードされたバイト値を使用する、可変長の文字列。最大文字列長はプラットフォームによって異なりますが、65535 バイトを超えることはできません。 **注意：** このフィールドに値を入力するときには、ユーザにはすべての文字が '*' を表示できます。
permutation	UTF-8 でエンコードされたバイト値を使用する、可変長の文字列。最大文字列長はプラットフォームによって異なりますが、65535 バイトを超えることはできません。 **注意：** このデータ型は、カスタム外部アダプタではサポートされません。
range	下限と上限を定義できる整数値。例： `<Parameter id="port" label="KDB Port" descr="IP port of the database listener" type="range" rangeLow="0" rangeHigh="65535" default="5001" use="required" />`
query	スタジオによってテーブル名から作成される文字列値。 **注意：** このデータ型は、カスタム外部アダプタではサポートされません。
runtimeDirectory	UTF-8 でエンコードされたバイト値を使用する、可変長の文字列。最大文字列長はプラットフォームによって異なりますが、65535 バイトを超えることはできません。 **注意：** このデータ型は、カスタム外部アダプタではサポートされません。
runtimeFilename	検出時のファイル名とは異なる場合の実行時のファイル名。 UTF-8 でエンコードされたバイト値を使用する、可変長の文字列で、最大文字列長はプラットフォームによって異なりますが、65535 バイトを超えることはできません。 **注意：** このデータ型は、カスタム外部アダプタではサポートされません。

データ型	説明
`string`	UTF-8 でエンコードされたバイト値を使用する、可変長の文字列。最大文字列長はプラットフォームによって異なりますが、65535 バイトを超えることはできません。
`tables`	アダプタで定義されている **getTables()** によって返される選択のリスト。
`text`	複数行テキストを格納可能な値。 **注意：** このデータ型は、カスタム外部アダプタではサポートされません。
`uint`	正の整数値。 有効な値の範囲は、0 ～ 0xffffffff です。

参照：
- *カスタム外部パラメータのデータ型* (571 ページ)
- *内部アダプタ* (21 ページ)
- *外部アダプタ* (136 ページ)
- *第 3 章、「カスタム・アダプタ」* (549 ページ)

入力アダプタ用の日付とタイムスタンプのフォーマット

日付とタイムスタンプのデータ型に関する多くのフォーマットがサポートされています。

以下の表の情報を使用して、日付とタイムスタンプのデータ型用のカスタム・フォーマットを作成します。

文字	説明
%a	ロケールの曜日名で表される曜日。 省略名または完全な名前のいずれかを指定できます。
%A	%a と同じ。
%b	ロケールの月名で表される月。 省略名または完全な名前のいずれかを指定できます。
%B	%b に同じ。
%c	ロケールの適切な日付と時間の表示。
%C	世紀を表す数字 (00 ～ 99)。 先頭にゼロを付加できますが、必須ではありません。

文字	説明
%d	日付 (01 ～ 31)。 先頭にゼロを付加できますが、必須ではありません。
%D	%m / %d / %y で表される日付。
%e	%d に同じ。
%h	%b に同じ。
%H	時 (24 時間表示) (00 ～ 23)。 先頭にゼロを付加できますが、必須ではありません。
%I	時 (12 時間表示) (01 ～ 12)。 先頭にゼロを付加できますが、必須ではありません。
%j	年間通し日付 (001 ～ 366)。 先頭にゼロを付加できますが、必須ではありません。
%m	月 (01 ～ 12)。 先頭にゼロを付加できますが、必須ではありません。
%M	分 (00 ～ 59)。 先頭にゼロを付加できますが、必須ではありません。
%n	任意のスペース。
%p	a.m. または p.m. のロケール相当。
%r	AM/PM 表記を使用する 12 時間での時刻。
%R	%H : %M として表される時刻。
%S	秒 (00 ～ 60)。 先頭にゼロを付加できますが、必須ではありません。
%t	任意のスペース。
%T	%H : %M : %S として表される時刻。
%U	年の何週目であるかを示す 10 進数 (00 ～ 53)。週は日曜日から始まります。 先頭にゼロを付加できますが、必須ではありません。
%w	10 進数 (0 ～ 6) で表される曜日。0 は日曜日を表します。 先頭にゼロを付加できますが、必須ではありません。
%W	年の何週目であるかを示す 10 進数 (00 ～ 53)。週は月曜日から始まります。 先頭にゼロを付加できますが、必須ではありません。
%x	ロケールの日付フォーマットで表される日付。
%X	ロケールの時刻フォーマットで表される時刻。
%y	世紀内の年。 世紀が指定されていない場合、69 ～ 99 の範囲の値は、それぞれ 1969 ～ 1999 の年と見なされ、00 ～ 68 の範囲の値はそれぞれ 2000 ～ 2068 の年と見なされます。 先頭にゼロを付加できますが、必須ではありません。

文字	説明
%Y	世紀も示す年。 たとえば、1988 です。
%%	% によって置換。

出力アダプタ用の日付とタイムスタンプのフォーマット

日付とタイムスタンプのデータ型に関する多くのフォーマットがサポートされています。

以下の表の情報を使用して、日付とタイムスタンプのデータ型用のカスタム・フォーマットを作成します。

文字	説明
%a	ロケールでの省略された曜日名。
%A	ロケールでの完全な曜日名。
%b	ロケールでの省略された月名。
%B	ロケールでの完全な月名。
%c	ロケールの適切な日付と時間の表示。
%C	年を 100 で除算し、10 進数 (00 〜 99) の整数にトランケートされた値。
%d	10 進数 (01 〜 31) で表される日付。
%D	%m / %d / %y と同じ。
%e	10 進数 (01 〜 31) で表される日付。 1 桁の場合は、先頭にスペースが付加されます。
%F	%Y - %m - %d に同じ。これは、ISO 8601:2000 標準日付フォーマットです。
%g	週に基づく年の下 2 桁で、10 進数 (00 〜 99) で表される。
%G	10 進数で表される、週に基づく年。 たとえば、1977 です。
%h	%b に同じ。
%H	10 進数 (00 〜 23) で表される時 (24 時間表示)。
%I	10 進数 (01 〜 12) で表される時 (12 時間表示)。
%j	10 進数 (001 〜 366) で表される年間通し日付。
%m	10 進数 (01 〜 12) で表される月。

文字	説明
%M	10 進数 (00 〜 59) で表される分。
%n	改行文字。
%p	a.m. または p.m. のいずれかのロケール相当。
%r	a.m. と p.m. で表記される時刻。
%R	24 時間表記の時刻 (%H : %M)。
%S	10 進数 (00 〜 60) で表される秒。
%t	タブ文字。
%T	%H : %M : %S として表される時刻。
%u	10 進数 (1 〜 7) で表される曜日。1 は月曜日を表します。
%U	年の何週目であるかを示す 10 進数 (00 〜 53)。 1 月の最初の日曜日が週 1 の最初の日で、この日より前の新年の日は、週 0 と見なされます。
%V	年の何週目であるかを示す 10 進数 (01 〜 53)。週は月曜日から始まります。 週に 1 月 1 日があり、4 日以上の新年の日がある場合、その週が週 1 と見なされます。その他の場合、1 月 1 日のある週は前年の最後の週と見なされ、翌週が週 1 と見なされます。 1 月 4 日と 1 月の最初の木曜日の両方は、常に週 1 です。
%w	10 進数 (0 〜 6) で表される曜日。0 は日曜日を表します。
%W	年の何週目であるかを示す 10 進数 (00 〜 53)。 1 月の最初の月曜日が週 1 の最初の日で、この日より前の新年の日は、週 0 と見なされます。
%x	ロケールの適切な日付の表示。
%X	ロケールの適切な時刻の表示。
%y	年の下 2 桁で、10 進数 (00 〜 99) で表される。
%Y	10 進数で表される年。 たとえば、1997 です。
%z	ISO 8601:2000 標準フォーマット (+hhmm または -hhmm) で表される UTC からのオフセット。タイム・ゾーンが判定できない場合、文字なしになります。 たとえば、"-0430" は、UTC から 4 時間 30 分遅れている (グリニッジの西側である) ことを意味します。
%Z	タイム・ゾーンの名前または略語。タイム・ゾーン情報が存在しない場合、文字なしになります。

文字	説明
%%	% によって置換。

第 2 章 Event Stream Processor でサポートされるアダプタ

Event Stream Processor は、さまざまな内部アダプタと外部アダプタをサポートします。

特に指定がないかぎり、これらのアダプタは、サーバとスタジオと同じプラットフォームとオペレーティング・システムをサポートします。 詳細については、『Event Stream Processor Installation Guide』を参照してください。

アダプタの概要

Event Stream Processor によってサポートされるアダプタの概要を以下に示します。以下の表には、アダプタのタイプ、アダプタを管理できるかどうか、スタジオで構成できるかどうか、保証された配信をサポートするかどうかが示されます。

アダプタ	タイプ	管理できる	スタジオで構成できる	保証された配信のサポート
AtomReader インプット	内部	可	可	不可
データベース・インプット	内部	可	可	不可
データベース・アウトプット	内部	可	可	不可
ファイル CSV インプット	内部	可	可	不可
ファイル CSV アウトプット	内部	可	可	不可
XML ファイル・インプット	内部	可	可	不可
XML ファイル・アウトプット	内部	可	可	不可
FIX ファイル・インプット	内部	可	可	不可
FIX ファイル・アウトプット	内部	可	可	不可
JMS CSV インプット	内部	可	可	可
JMS CSV アウトプット	内部	可	可	可

アダプタ	タイプ	管理できる	スタジオで構成できる	保証された配信のサポート
JMS カスタム・インプット	内部	可	可	可
JMS カスタム・アウトプット	内部	可	可	可
JMS FIX インプット	内部	可	可	可
JMS FIX アウトプット	内部	可	可	可
JMS Object Array インプット	内部	可	可	可
JMS Object Array アウトプット	内部	可	可	可
JMS XML インプット	内部	可	可	可
JMS XML アウトプット	内部	可	可	可
ランダム・タプル生成インプット	内部	可	可	不可
ソケット FIX インプット	内部	可	可	不可
ソケット FIX アウトプット	内部	可	可	不可
ソケット (クライアント側) CSV インプット	内部	可	可	不可
ソケット (クライアント側) CSV アウトプット	内部	可	可	不可
ソケット (クライアント側) XML インプット	内部	可	可	不可
ソケット (クライアント側) XML アウトプット	内部	可	可	不可
ソケット (サーバ側) XML インプット	内部	可	可	不可
ソケット (サーバ側) XML アウトプット	内部	可	可	不可
ソケット (サーバ側) CSV インプット	内部	可	可	不可
ソケット (サーバ側) CSV アウトプット	内部	可	可	不可
SMTP アウトプット	内部	可	可	不可
Sybase IQ アウトプット	内部	可	可	不可
WebSphere MQ インプット	内部	可	可	可

アダプタ	タイプ	管理できる	スタジオで構成できる	保証された配信のサポート
WebSphere MQ アウトプット	内部	可	可	可
FIX インプット	外部	可	可	不可
FIX アウトプット	外部	可	可	不可
フレックス・アウトプット	外部	不可	不可	不可
HTTP アウトプット	外部	可	可	不可
KDB インプット	外部	可	可	不可
KDB アウトプット	外部	可	可	不可
ログファイル・インプット	外部	不可	不可	不可
NYSE Technologies インプット	外部	可	可	不可
オープン・インプット	外部	不可	不可	不可
オープン・アウトプット	外部	不可	不可	不可
RAP アウトプット	外部	不可	不可	不可
ロイター・マーケットフィード・インプット	外部	不可	不可	不可
ロイター・マーケットフィード・アウトプット	外部	不可	不可	不可
ロイター OMM インプット	外部	不可	不可	不可
ロイター OMM アウトプット	外部	不可	不可	不可
RTView アウトプット	外部	不可	不可	不可
Tibco Rendezvous インプット	外部	可	不可	可
Tibco Rendezvous アウトプット	外部	可	不可	可

アダプタ・プロパティ・セットの編集

スタジオで CCR プロジェクト設定エディタを使用して、アダプタ・プロパティ・セットを設定します。 プロパティ・セットはプロパティの再使用可能なセットで、プロジェクト設定ファイルに格納されます。

プロパティ・セットはツリー形式で表示され、個々のプロパティ定義は、プロパティ・セットの子として表示されます。

1. CCR プロジェクト設定エディタで、[Adapter Properties] タブを選択します。
2. 新しいアダプタ・プロパティ・ノードを作成するには、[Add] をクリックします。
3. [Property Set Details] ペインでプロパティ・ノードの名前を定義します。
4. 新しいプロパティをプロパティ・セットに追加するには、セットを右クリックし、[New] > [Property] を選択します。

   注意： プロパティ項目は必要な数だけプロパティ・セットに追加できます。
5. プロパティを設定するには、次の手順に従います。
   a) [Property Details] ペインでプロパティの名前を定義します。
   b) プロパティの値を入力します。
6. (オプション) プロパティ値を暗号化するには、次の手順に従います。
   a) プロパティ値を選択して、[Encrypt] をクリックします。
   b) [Cluster URI] やクレデンシャル・フィールドなどの必須フィールドを入力します。
   c) [Encrypt] をクリックします。
      値とそれに関連するフィールドには、ランダム化された暗号文字が設定されます。

      注意： 暗号化をリセットするには、該当するフィールドの隣にある [Encrypt] をクリックします。 必要に応じて値を変更し、[Reset] をクリックします。
7. [All Adapter Properties] リストから項目を削除するには、次の手順に従います。
   - プロパティ・セットを右クリックし、[Remove] を選択する。または
   - プロパティを右クリックして、[Delete] を選択する。

内部アダプタ

Event Stream Processor には、標準のデータ・フォーマットを処理するために内部アダプタが用意されています。

内部アダプタは、Event Stream Processor 内で実行します。 サーバは、クエリ・モジュールの実行を介して、これらのアダプタの起動と停止を実行します。

参照：
- *アダプタ・パラメータのデータ型* (9 ページ)
- *スキーマ検出をサポートするアダプタ* (585 ページ)

AtomReader インプット・アダプタ

アダプタのタイプ： atomreader_in。AtomReader インプット・アダプタを使用して、ATOM データソースから情報を受信できます。

ATOM データソースは、URL での接続が可能で、特化した XML フォーマットで情報を転送します。 アダプタが ATOM データソースから受信した情報は、Event Stream Processor ストリームに挿入されます。

受信 XML 情報に、

- feed_title
- feed_link
- feed_author_name
- entry_title
- entry_link
- entry_content

があることを確認します。

注意： アダプタは、XML ファイルの他のすべてのフィールドを無視します。

ATOM で使用される特別な XML フォーマットの詳細については、次を参照してください。 http://www.atomenabled.org/

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは atomreader_in です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Source URL	**URL**	`string`	(必須) ATOM データソースの URL。
Refresh interval	**refreshInterval**	`interval`	(オプション) 指定された URL に対してデータを取得するためにクエリを発行する頻度を決定。 間隔フォーマットで修飾されていないかぎり、アダプタは間隔をマイクロ秒単位で測定します。 デフォルト値は 60000 ミリ秒です。
Timestamp format	**timestampFormat**	`string`	(詳細) タイムスタンプ値を解析するためのフォーマット文字列。 デフォルト値は、%Y-%m-%dT%H:%M:%S です。
Date Format	**dateFormat**	`string`	(詳細) 日付値を解析するためのフォーマット文字列。 デフォルト値は、%Y-%m-%dT%H:%M:%S です。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

データベース・アダプタ

Event Stream Processor には、データベース・インプット・アダプタとデータベース・アウトプット・アダプタが用意されています。 データベース・インプット・アダプタはデータベース・テーブルからデータを受け取り、データベース・アウトプット・アダプタはデータベース・テーブルにデータを送ります。

いくつかの異なる JDBC ドライバと ODBC ドライバは、データベース・アダプタと共に使用できます。 ODBC ドライバを使用するには、ドライバ・マネージャがインストールされていることを確認してください。

データベース・インプット・アダプタ

アダプタのタイプ： db_in。データベース・インプット・アダプタは、データベース・テーブルからデータを受け取ります。

アダプタを使用して、定期的にテーブルをポーリングして更新を受け取れます。必須プロパティは、接続しているデータベース・タイプによって決まります。JDBC でサポートされているデータベースは、Adaptive Server® Enterprise、Microsoft SQL Server、IBM DB2、Oracle、KDB です。 ODBC でサポートされているデータベースは、Adaptive Server Enterprise、Microsoft SQL Server、IBM DB2、Oracle、Sybase IQ、SQL Anywhere®、TimesTen、MySQL 5.x、PostgreSQL です。

`service.xml` ファイルには、サービス定義と、データベース接続に必要なプロパティがあります。 サービス定義名については、`service.xml` ファイルの設定と保守の担当者に問い合わせてください。 `service.xml` ファイルを使用したデータベース接続の設定については、『管理者ガイド』を参照してください。

テーブル選択を上書きし、任意のクエリからデータを取得するには、**query** プロパティを使用します。 このアダプタは、スキーマ検出をサポートします。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは db_in です。

重要： ASE データベースでは、スキーマ検出使用時にすべてのテーブルを検出するには、テンポラリ・データベース (tempdb) で "ddl in tran" オプションを有効にします。 次に、tempdb でチェックポイントを実施するか、またはデータベース・インスタンスを再起動して、サーバを更新します。 "ddl in tran" オプションの詳細については、Adaptive Server のマニュアルを参照してください。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Database Service	**service**	`string`	(必須) `service.xml` ファイルで定義されているデータベース・サービスの名前。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Database Query	**query**	`string`	(オプション) アダプタによって実行される SQL クエリ。 デフォルト値はありません。 **注意：** アダプタ定義で、**query** または **table** を定義する必要があります。 両方のパラメータが定義されている場合は、**query** パラメータが使用されます。
Input Table Name	**table**	`tables`	(オプション) 読み込むテーブルの名前。 デフォルト値はありません。
Poll Period (in seconds)	**pollperiod**	`uint`	(詳細) 新しいコンテンツをポーリングする、秒単位の間隔。 デフォルト値は 0 で、ポーリングしないことを示します。
Date Format	**dateFormat**	`string`	(詳細) 日付値を解析するためのフォーマット文字列。 デフォルト値は、`%Y-%m-%dT%H:%M:%S` です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプ値を解析するためのフォーマット文字列。 デフォルト値は、`%Y-%m-%dT%H:%M:%S` です。

プロパティ・ラベル	プロパティ ID	タイプ	説明

Field Mapping	**permutation**	`permutation`	(詳細) プラットフォーム固有のフィールドと外部フィールドとの間のマッピング。 デフォルト値はありません。
			注意：
			• Oracle 11g と DB2 9.7 では、メタデータ・サービスが結果を大文字で返すので、permutation のデータベース・カラム・キーは大文字にする必要があります。 たとえば、CCL では、次のコマンドは機能しません。
			`permutation= 'Subject=subject:c_string =c_string'`
			しかし、次のコマンドは機能します。
			`permutation= 'Subject=SUBJECT:c_string =C_STRING'`
			• ASE 15.5 と SQL Server 2008 では、**SELECT** 文で名前を変更しないかぎり、メタデータ・サービスの結果は、定義されているカラム名の大文字／小文字と同じです。

PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

データベース・インプット・アダプタには、次の制限があります。

- ポーリングを行う場合は、これが唯一のアダプタであることを確認する。
- 次回のポーリング時に、他の任意のソースから受け取ったデータ更新は行われない。

データベース・アウトプット・アダプタ

アダプタのタイプ： db_out。 データベース・アウトプット・アダプタは、データベース・テーブルにデータを送ります。

truncateTable プロパティを使用して、アダプタの起動時にテーブルをトランケートできます。 必須プロパティは、接続しているデータベース・タイプによって決まります。 JDBC でサポートされているデータベースは、Adaptive Server Enterprise、Microsoft SQL Server、IBM DB2、Oracle、KDB です。 ODBC でサポートされているデータベースは、Adaptive Server Enterprise、Microsoft SQL Server、IBM DB2、Oracle、Sybase IQ、SQL Anywhere、TimesTen、MySQL 5.x、PostgreSQL です。

`service.xml` ファイルには、サービス定義と、データベース接続に必要なプロパティがあります。 サービス定義名については、`service.xml` ファイルの設定と保守の担当者に問い合わせてください。 `service.xml` ファイルを使用したデータベース接続の設定の詳細については、『管理者ガイド』を参照してください。

注意： Oracle ODBC ドライバは、SQL_C_SBIGINT パラメータと SQL_C_UBIGINT パラメータをサポートしません。このため、データベース・アウトプット・アダプタが `long` や `interval` の Event Stream Processor データ型を `bigint` データ型のカラムに書き込もうとすると、エラーが発生します。 Oracle ODBC ドライバや TimesTen ODBC ドライバをデータベース・アウトプット・アダプタと共に正常に使用するには、このパラメータ "`<Parameter Name ="WriteBigIntAsChar" > true < /Parameter >`" を `service.xml` ファイルに追加します。

異なる日付フォーマットを指定する例は、日付カラムを Oracle Date カラムに挿入する場合です。 Oracle のデフォルトの日付フォーマットは、 `04-Apr-1964 17:12:00` です。したがって、**dateFormat** パラメータを `d-%b-%Y %H:%M:%S` と指定します。

重要： データベース・アウトプット・アダプタが ODBC ドライバを使用して PostgreSQL データベースに正常に書き込めるようにするには、ODBC 設定で "Server side prepare" オプションを有効にします。 このオプションの詳細については、ODBC のドキュメントを参照してください。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは db_out です。

重要： ASE データベースでは、スキーマ検出使用時にすべてのテーブルを検出するには、テンポラリ・データベース (tempdb) で "ddl in tran" オプションを有効にします。 次に、tempdb でチェックポイントを実施するか、またはデータベース・インスタンスを再起動して、サーバを更新します。 "ddl in tran" オプションの詳細については、Adaptive Server のマニュアルを参照してください。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Database Service	**service**	`string`	(必須) `service.xml` ファイルで定義されているデータベース・サービスの名前。 デフォルト値はありません。
Date Format	**dateFormat**	`string`	(詳細) 日付値を解析するためのフォーマット文字列。 デフォルト値は、`%Y-%m-%dT%H:%M:%S` です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプ値を解析するためのフォーマット文字列。 デフォルト値は、`%Y-%m-%dT%H:%M:%S` です。

プロパティ・ラベル	プロパティ ID	タイプ	説明

Field Mapping	**permutation**	`permutation`	(詳細) プラットフォーム固有のフィールドと外部フィールドとの間のマッピング。 デフォルト値はありません。
			注意：
			• Oracle 11g と DB2 9.7 では、メタデータ・サービスが結果を大文字で返すので、permutation のデータベース・カラム・キーは大文字にする必要があります。 たとえば、CCL では、次のコマンドは機能しません。
			`permutation= 'Subject=subject:c_string =c_string'`
			しかし、次のコマンドは機能します。
			`permutation= 'Subject=SUBJECT:c_string =C_STRING'`
			• ASE 15.5 と SQL Server 2008 では、**SELECT** 文で名前を変更しないかぎり、メタデータ・サービスの結果は、定義されているカラム名の大文字／小文字と同じです。

Only Base Content	**onlyBase**	`boolean`	(詳細) ストリームの初期内容を 1 回送信する。デフォルト値は false です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Batch Limit	**batchLimit**	`uint`	(詳細) バッチとして処理するレコードの数。 デフォルト値は、1 です。 **注意：** バッチ処理で UPSERT を使用すると、アダプタが削除を受け取った場合にこの処理が終了することがあるので、パフォーマンスが悪影響を受ける可能性があります。 UPSERT はストリームの内容に基づいて INSERT または UPDATE に解釈されるので、これらの操作のグルーピングが難しくなります。 しかも、操作 (INSERT、UPDATE、DELETE、UPSET) を頻繁に変更すると、バッチ処理使用の最適化度が低くなります。
Data Location	**datalocation**	`string`	(詳細) プロジェクト設定のプロパティを検索するロケーション。 デフォルト値はありません。
Output Table Name (runtime)	**table**	`tables`	(オプション) データの書き込み先のテーブルの名前。 デフォルト値はありません。
Include Base Content	**outputBase**	`boolean`	(オプション) ストリームの更新内容に加えて、ストリームの初期内容を出力する。デフォルト値は false です。
Truncate the Database Table	**truncateTable**	`boolean`	(オプション) データベース・テーブルをトランケートすることによって開始し、次にストリーミング・データを設定する。デフォルト値は false です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

データベース・アウトプット・アダプタには、次の制限があります。

- 出力テーブルが存在する必要がある。
- 各ローが SQL 文に翻訳されるので、更新が低速である。
- メモリ・ストアを使用している場合、ストリーム内のデータに対して UPSERT、UPDATE、DELETE のみを実施できる。

データベース・アダプタのデータ型のマッピング

Event Stream Processor データ型と Sybase Adaptive Service Enterprise、Microsoft SQL Server、IBM DB2、Oracle、KDB のデータ型のマッピング。

データ型マッピング：Sybase ASE

Event Stream Processor のデータ型と Adaptive Server Enterprise 15.5 のデータ型のマッピング。

Event Stream Processor のデータ型	Adaptive Server Enterprise のデータ型
`integer`	`int`
`long`	`bigint`
`float`	`float`
`date`	`datetime`
`string`	`varchar(n)`
`money`	`money`
`timestamp`	`bigdatetime`

Event Stream Processor のデータ型	Adaptive Server Enterprise のデータ型
boolean	smallint または bit (null 値は未サポート)
money1	numeric(19,1)
money2	numeric(19,2)
money3	numeric(19,3)
money4	numeric(19,4)
money5	numeric(19,5)
money6	numeric(19,6)
money7	numeric(19,7)
money8	numeric(19,8)
money9	numeric(19,9)
money10	numeric(19,10)
money11	numeric(19,11)
money12	numeric(19,12)
money13	numeric(19,13)
money14	numeric(19,14)
money15	numeric(19,15)
interval	bigint
bigdatetime	bigdatetime
binary	varbinary(n)

データ型マッピング：Microsoft SQL Server データベース

Event Stream Processor のデータ型と Microsoft SQL Server 2008 R2 のデータ型のマッピング。

Event Stream Processor のデータ型	SQL のデータ型
integer	int

Event Stream Processor のデータ型	SQL のデータ型
long	bigint
float	float
date	datetime
string	varchar(n)
money	money
timestamp	datetime2
boolean	smallint または bit (null 値は未サポート)
money1	numeric(19,1)
money2	numeric(19,2)
money3	numeric(19,3)
money4	numeric(19,4)
money5	numeric(19,5)
money6	numeric(19,6)
money7	numeric(19,7)
money8	numeric(19,8)
money9	numeric(19,9)
money10	numeric(19,10)
money11	numeric(19,11)
money12	numeric(19,12)
money13	numeric(19,13)
money14	numeric(19,14)
money15	numeric(19,15)
interval	bigint

Event Stream Processor のデータ型	SQL のデータ型
bigdatetime	datetime2
binary	varbinary(n)

データ型マッピング：IBM DB2 データベース
Event Stream Processor のデータ型と IBM DB2 9.7 のデータ型のマッピング。

Event Stream Processor のデータ型	IBM DB2 のデータ型
integer	int
long	bigint
float	float
date	timestamp
string	varchar(n)
money	decimal(19,5)
timestamp	timestamp
boolean	smallint または bit (null 値は未サポート)
money1	decimal(19,1)
money2	decimal(19,2)
money3	decimal(19,3)
money4	decimal(19,4)
money5	decimal(19,5)
money6	decimal(19,6)
money7	decimal(19,7)
money8	decimal(19,8)
money9	decimal(19,9)
money10	decimal(19,10)

Event Stream Processor のデータ型	IBM DB2 のデータ型
money11	decimal(19,11)
money12	decimal(19,12)
money13	decimal(19,13)
money14	decimal(19,14)
money15	decimal(19,15)
interval	bigint
bigdatetime	timestamp
binary	blob

データ型マッピング：Oracle データベース

Event Stream Processor のデータ型と Oracle 11g のデータ型のマッピング。

Event Stream Processor のデータ型	Oracle のデータ型
integer	int
long	number(19)
float	float
date	date
string	varchar2(n)
money	number(19,4)
timestamp	timestamp
boolean	smallint または bit (null 値は未サポート)
money1	number(19,1)
money2	number(19,2)
money3	number(19,3)
money4	number(19,4)

Event Stream Processor のデータ型	Oracle のデータ型
money5	number(19,5)
money6	number(19,6)
money7	number(19,7)
money8	number(19,8)
money9	number(19,9)
money10	number(19,10)
money11	number(19,11)
money12	number(19,12)
money13	number(19,13)
money14	number(19,14)
money15	number(19,15)
interval	number(19)
bigdatetime	timestamp
binary	blob

データ型マッピング：KDB データベース
Event Stream Processor のデータ型と KDB のデータ型のマッピング。

Event Stream Processor のデータ型	KDB データ型
integer	int
long	long
float	float
date	datetime
string	symbol
money	–
timestamp	datetime
boolean	boolean (null は未サポート)

Event Stream Processor のデータ型	KDB データ型
money1	–
money2	–
money3	–
money4	–
money5	–
money6	–
money7	–
money8	–
money9	–
money10	–
money11	–
money12	–
money13	–
money14	–
money15	–
interval	long
bigdatetime	–
binary	–

注意： 出力に構成する Event Stream Processor のデータ型は、等価のデータ型が KDB にあるものだけにします。

ファイル CSV インプット・アダプタ

アダプタのタイプ： dsv_in。ファイル CSV インプット・アダプタは、Event Stream Processor の区切られたフォーマットでファイルを読み込みます。

このアダプタを使用して、データ・ファイルに追加された新規データをポーリングします。 このファイルは、ヘッダを必要としません。 ファイルにヘッダが含まれる場合、そのヘッダはフィールド名を指定します。

次は、データ・ファイルのレコード・フォーマット例です。

```
1. hasHeader=true
delimiter=,
expectStreamNameOpcode=false

Ts,ItemID,Price,Quantity,WarehouseZipCode,DeliveryZipCode
2004/06/17 10:00:00.000000,SKU1276532,50.00,1,10012,94086
2004/06/17 10:00:05.000000,SKU6723143,23.00,2,10012,94043

2. expectStreamNameOpcode=true
delimiter=,

Trades_in,i,2004/06/17
10:00:00.000000,SKU1276532,50.00,1,10012,94086
Trades_in,i,2004/06/17
10:00:05.000000,SKU6723143,23.00,2,10012,94043

3. expectStreamNameOpcode=false
timestampFormat=%Y/%m/%d %H:%M:%S
delimiter=,

2004/06/17 10:00:00.000000,SKU1276532,50.00,1,10012,94086
2004/06/17 10:00:05.000000,SKU6723143,23.00,2,10012,94043
```

このアダプタは、スキーマ検出をサポートします。CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは dsv_in です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Directory	**dir**	`directory`	(必須) アダプタが読み込むデータ・ファイルの絶対パスを指定。たとえば、`<username>/<folder name>` のように記述します。デフォルト値はありません。
File (in Directory)	**file**	`tables`	(必須) 読み込むファイル。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Stream name, opcode expected	**expectStreamNameOpcode**	`boolean`	(オプション) true の場合、アダプタは最初の 2 つのフィールドをそれぞれストリーム名と opcode と解釈する。ストリーム名に一致しないメッセージは、破棄されます。デフォルト値は false です。
Field Count	**fieldCount**	`uint`	(オプション) ソース・ストリームの値と異なる場合、CSV ファイルのフィールドの数。デフォルト値は 0 です。
Repeat Count	**repeatCount**	`uint`	(オプション) 入力データが繰り返される回数。 -1 に設定された場合、入力データは無制限に繰り返します。デフォルト値は 0 です。 **注意：** このパラメータは、連続ストリーム・ソースをテストするために使用できます。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Repeat Field	**repeatField**	`string`	(オプション) 各繰り返しで、どの数値フィールドの値を増加するかを決定する。 デフォルト値はハイフン (-) です。 **注意：** • **repeatCount** にゼロ以外の値が設定されている場合、ストリーム・カラム名を指定する。 • repeatColumn がストリームのキー・カラムの場合、入力ファイルで複数のローを指定するときに重複が発生しないようにする。 • アダプタがウィンドウに付加されている場合、**repeatField** はキー・カラムである必要がある。
Delimiter	**delimiter**	`string`	(詳細) カラムを区切るために使用する記号。デフォルト値は、カンマ (,) です。
Has Header	**hasHeader**	`boolean`	(詳細) 最初の行にフィールドの説明があるかどうかを決定する。デフォルト値は false です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Directory (runtime)	**runtimeDir**	`runtimeDirectory`	(詳細) 検出時に定義されたロケーションと値が異なる場合、実行時のデータ・ファイルのロケーション。 デフォルト値はありません。
File Pattern	**filePattern**	`string`	(詳細) 検出対象のファイルを検索するために使用するパターン。 デフォルト値は、`*.csv` です。
Poll Period (秒)	**pollperiod**	`uint`	(詳細) 新しいコンテンツをポーリングする、秒単位の間隔。 0 に設定された場合、ファイル CSV インプット・アダプタは、レコードに追加されたファイルをポーリングしません。デフォルト値は 0 です。
Convert to Safe Opcodes	**safeOps**	`boolean`	(詳細) opcode の INSERT と UPDATE を UPSERT に変換し、DELETE を SAFEDELETE に変換する。デフォルト値は false です。
Skip Deletes	**skipDels**	`boolean`	(詳細) DELETE または SAFEDELETE の opcode があるローをスキップする。デフォルト値は false です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Date Format	**dateFormat**	`string`	(詳細) 日付値を解析するためのフォーマット文字列。 デフォルト値は、`%Y-%m-%dT%H:%M:%S` です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプ値を解析するためのフォーマット文字列。 デフォルト値は、`%Y-%m-%dT%H:%M:%S` です。
Block Size	**blockSize**	`int`	(詳細) 擬似トランザクションになる、ブロックあたりのレコード数。デフォルト値は 1 です。
Field Mapping	**permutation**	`permutation`	(詳細) プラットフォーム固有のフィールドと外部フィールドとの間のマッピング。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

ファイル CSV インプット・アダプタには、以下の 3 つの制限があります。

- ポーリングするときに、ファイルに追加できるが、ファイルを上書きしたり、置き換えたりできない。 ファイル・ローのストリーム名は無視され、すべてのデータは同じストリームに送信されます。
- 検索が正しく機能するために、デリミタ文字とヘッダ存在フラグを設定して、実際のデータに一致させる。
- 同じディレクトリ内で、異なるデリミタ文字を使用するファイル、またはヘッダの有無が異なるファイルを混在させない。 間違ったデリミタまたはヘッダを使用するファイルは正しく検出されません。

ファイル CSV アウトプット・アダプタ

アダプタのタイプ： dsv_out。 ファイル CSV アウトプット・アダプタは、Event Stream Processor の区切られたフォーマットのファイルとしてデータを書き込みます。

このファイルは、ヘッダを必要としません。 ファイルにヘッダが含まれる場合、そのヘッダはフィールド名を指定します。 このアダプタは、スキーマ検出をサポートしません。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは dsv_out です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Directory	**dir**	`directory`	(必須) アダプタが読み込むデータ・ファイルの絶対パスを指定。たとえば、`<username>/<folder name>` のように記述します。デフォルト値はありません。
File (in Directory)	**file**	`tables`	(必須) アダプタがデータを書き込むファイル。デフォルト値はありません。
Include Base Content	**outputBase**	`boolean`	(オプション) ストリームの更新内容だけでなく、初期内容も記録する。デフォルト値は false です。
Only Base Content	**onlyBase**	`boolean`	(オプション) ストリームの初期内容のワンタイム・スナップショットを送信する。デフォルト値は false です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Prepend StreamNameOpcode	**prependStreamNameOpcode**	`boolean`	(オプション) true の場合、各マージはストリーム名と opcode で開始する。デフォルト値は false です。
Delimiter	**delimiter**	`string`	(詳細) カラムを区切るために使用する記号。 デフォルト値は、カンマ (,) です。
Has Header	**hasHeader**	`boolean`	(詳細) 最初の行がフィールドの説明を含むかどうかを決定する。デフォルト値は false です。
Directory (runtime)	**runtimeDir**	`runtimeDirectory`	(詳細) 実行時のデータ・ファイルのロケーション (検出時と異なる場合)。 デフォルト値はありません。
File Pattern	**filePattern**	`string`	(詳細) 検出対象のファイルを検索するために使用するパターン。 デフォルト値は、`*.csv` です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Date Format	**dateFormat**	`string`	(詳細) データ値を解析するためのフォーマット文字列。 デフォルト値は、`%Y-%m-%dT%H:%M:%S` です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプ値を解析するための フォーマット文字列。 デフォルト値は、`%Y-%m-%dT%H:%M:%S` です。
Field Mapping	**permutation**	`permutation`	(詳細) プラットフォーム固有のフィールドと外部フィールドとの間のマッピング。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

XML ファイル・インプット・アダプタ

アダプタのタイプ： xml_in。XML ファイル・インプット・アダプタは、XML フォーマットでファイルを読み込みます。

このアダプタは、ファイルに追加された新規データをポーリングし、スキーマ検出をサポートします。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは xml_in です。

次は、データ・ファイルのレコード・フォーマット例です。

```
<Trades Id="0" Symbol="EBAY" TradeTime="2000-05-04T12:00:00"
Price="140.0" Shares="50" />
<Trades Id="1" Symbol="EBAY" TradeTime="2000-05-04T12:00:01"
Price="150.0" Shares="500" />
```

プロパティ・ラベル	プロパティ ID	タイプ	説明
Directory	**dir**	`directory`	(必須) アダプタが読み込むデータ・ファイルの絶対パスを指定。たとえば、`<username>/<folder name>` のように記述します。 デフォルト値はありません。
File (in Directory)	**file**	`tables`	(必須) アダプタがデータを書き込むファイル。 デフォルト値はありません。
Match Stream Name	**matchStreamName**	`boolean`	(オプション) true の場合、XML 要素名がストリーム名に対してマッチングされる。一致しないメッセージは、破棄されます。デフォルト値は false です。
Repeat Count	**repeatCount**	`uint`	(オプション) 入力データが繰り返される回数。 -1 に設定された場合、入力データは無制限に繰り返します。 デフォルト値は 0 です。 **注意：** このパラメータは、連続ストリーム・ソースをテストするために使用できます。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Repeat Field	**repeatField**	`string`	(オプション) 各繰り返しで、どの数値フィールドの値を増加するかを決定する。 デフォルト値はハイフン (-) です。 **注意：** • **repeatCount** にゼロ以外の値が設定されている場合、ストリーム・カラム名を指定する。 • repeatColumn がストリームのキー・カラムの場合、入力ファイルで複数のローを指定するときに重複が発生しないようにする。 • アダプタがウィンドウに付加されている場合、**repeatField** はキー・カラムである必要がある。
Directory (run-time)	**runtimeDir**	`runtimeDirectory`	(詳細) 実行時のデータ・ファイルのロケーション (検出時と異なる場合)。 デフォルト値はありません。
File Pattern	**filePattern**	`string`	(詳細) 検出対象のファイルを検索するために使用するパターン。 デフォルト値は、`*.xml` です。
Poll Period (秒)	**pollperiod**	`uint`	(詳細) 新しいコンテンツをポーリングする、秒単位の間隔。デフォルト値は 0 です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Convert to Safe Opcodes	**safeOps**	`boolean`	(詳細) opcode の INSERT と UPDATE を UPSERT に変換する。 DELETE を SAFEDELETE に変換します。 デフォルト値は false です。
Skip Deletes	**skipDels**	`boolean`	(詳細) DELETE または SAFEDELETE の opcode を含むローをスキップする。デフォルト値は false です。
Date Format	**dateFormat**	`string`	(詳細) データ値を解析するためのフォーマット文字列。 デフォルト値は、%Y-%m-%dT%H:%M:%S です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプ値を解析するためのフォーマット文字列。 デフォルト値は、%Y-%m-%dT%H:%M:%S です。
Block Size	**blockSize**	`int`	(詳細) 擬似トランザクションになる、ブロックあたりのレコード数を決定する。デフォルト値は 1 です。
Field Mapping	**permutation**	`permutation`	(詳細) プラットフォーム固有のフィールドと外部フィールドとの間のマッピング。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

既知の制限事項：

- ポーリング時に、ファイルに追加できるが、そのファイルを上書きまたは置き換えることはできない。
- ファイル・エントリのストリーム名は無視される。
- 同じディレクトリにデータ・ファイルとモデル XML ファイルを混在させない。 混在させると、Event Stream Processor XML ファイルが無効として検出されます。

XML ファイル・アウトプット・アダプタ

アダプタのタイプ： xml_out。 XML ファイル・アウトプット・アダプタは、XML フォーマットのファイルとしてデータを書き込みます。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは xml_out です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Directory	**dir**	`directory`	(必須) アダプタが読み込むデータ・ファイルの絶対パスを指定。たとえば、`<username>/<folder name>` のように記述します。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
File (in Directory)	**file**	`tables`	(必須) アダプタがデータを書き込むファイル。 デフォルト値はありません。
Include Base Content	**outputBase**	`boolean`	(オプション) ストリームの更新内容だけでなく、初期内容も記録する。デフォルト値は false です。
Only Base Content	**onlyBase**	`boolean`	(オプション) ストリームの初期内容のワンタイム・スナップショットを送信する。デフォルト値は false です。
Directory (run-time)	**runtimeDir**	`runtimeDirectory`	(詳細) 実行時のデータ・ファイルのロケーション (検出時と異なる場合)。 デフォルト値はありません。
File Pattern	**filePattern**	`string`	(詳細) 検出対象のファイルを検索するために使用するパターン。 デフォルト値は、`*.xml` です。
Date Format	**dateFormat**	`string`	(詳細) データ値を解析するためのフォーマット文字列。 デフォルト値は、`%Y-%m-%dT%H:%M:%S` です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプ値を解析するためのフォーマット文字列。 デフォルト値は、`%Y-%m-%dT%H:%M:%S` です。
Field Mapping	**permutation**	`permutation`	(詳細) プラットフォーム固有のフィールドと外部フィールドとの間のマッピング。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

FIX ファイル・インプット・アダプタ

Adapter type: fixfile_in。FIX ファイル・インプット・アダプタは、FIX メッセージを読み込み、ストリーム・レコードとして書き込みます。

各ストリームは、特定のタイプの FIX メッセージをホストします。 アダプタは、他のいずれの FIX タイプのメッセージも無視します。 アダプタは、以下を除くすべての FIX フィールドをストリームのカラムと同じ順序で書き込みます。

- BeginString
- BodyLength
- MsgType
- CheckSum

ストリーム・カラムの名前が、FIX プロトコルの仕様に対応していることを確認します。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは fixfile_in です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
FIX Version	**fixVersion**	`choice`	(必須) FIX プロトコルのバージョン。デフォルト値は 4.2 です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
FIX Message Type	**fixMessageType**	`string`	(必須) ストリームによってホストされるメッセージのタイプ。 デフォルト値はありません。
File	**fileName**	`filename`	(必須) FIX メッセージのある入力ファイルへのパス。 デフォルト値はありません。
Date Format	**dateFormat**	`string`	(詳細) 日付のフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプのフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

既知の制限事項：

- このアダプタは、完全な FIX エンジンではない。
- FIX のバージョン 4.2、4.3、4.4、5.0 のみをサポートする。
- グループとコンポーネントの繰り返しはサポートされない。
- INSERT opcode のみをサポートする。

参照：

- *FIX アダプタ* (147 ページ)

FIX ファイル・インプット・アダプタのデータ型のマッピング

Event Stream Processor のデータ型は、FIX データ型にマップされます。

Event Stream Processor のデータ型	QuickFix のデータ型
integer	boolean
string	byte[]
string	char
string	string
date	date
float	float
integer	integer
date または timestamp	UTCDateOnly
date または timestamp	UTCTimeOnly
date または timestamp	UTCTimeStamp

FIX ファイル・アウトプット・アダプタ

アダプタのタイプ： fixfile_out。 FIX ファイル・アウトプット・アダプタは、ストリーム・データを FIX メッセージとしてファイルに書き込みます。

各ストリームは、特定のタイプの FIX メッセージをホストします。 アダプタは、メッセージを改行なしで隣接してファイルに書き込みます。 次の FIX フィールドが生成されます。

- BeginString
- BodyLength
- MsgType
- CheckSum

残りのフィールドをストリーム・カラムの適切な順序で書き込みます。 ストリーム・カラムの名前が、FIX プロトコルの仕様に対応していることを確認します。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは fixfile_out です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
FIX Version	**fixVersion**	`choice`	(必須) FIX プロトコルのバージョン。デフォルト値は 4.2 です。
FIX Message Type	**fixMessageType**	`string`	(必須) ストリームによってホストされるメッセージのタイプ。 デフォルト値はありません。
File	**fileName**	`filename`	(必須) FIX メッセージのある入力ファイルへのパス。 デフォルト値はありません。
Date Format	**dateFormat**	`string`	(詳細) 日付のフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプのフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

既知の制限事項：

- このアダプタは、完全な FIX エンジンではない。
- FIX のバージョン 4.2、4.3、4.4、5.0 のみをサポートする。
- グループとコンポーネントの繰り返しはサポートされない。
- スキーマ検出はサポートされない。
- INSERT opcode のみをサポートする。

参照：

- *FIX アダプタ* (147 ページ)

FIX ファイル・アウトプット・アダプタのデータ型のマッピング

Event Stream Processor のデータ型は、FIX データ型にマップされます。

Event Stream Processor のデータ型	QuickFix のデータ型
integer	boolean
string	byte[]
string	char
string	string
date	date
float	float
integer	integer
date または timestamp	UTCDateOnly
date または timestamp	UTCTimeOnly
date または timestamp	UTCTimeStamp

JMS アダプタ

Event Stream Processor には、次の 5 つの JMS インプット・アダプタと JMS アウトプット・アダプタが用意されています。 CSV、カスタム、FIX、Object Array、XML です。

JMS アダプタ向けキューイング・システムの設定

JMS アダプタを使用するには、ネーム・サーバを使用するキューイング・システムを設定します。

前提条件

JNDI ネーム・サーバをサポートするキューイング・システム。

手順

1. ネーム・サーバをセットアップします。
   一部のキューイング・システムには、JMS アダプタに接続できる内部ネーム・サーバがあります。
2. JMS オブジェクトを管理するネーム・サーバを使用するために、キューイング・システムをセットアップします。

詳細については、お使いのサードパーティ製キューイング・システムと共に提供されているマニュアルを参照してください。

3. JNDI コンテキスト・ライブラリとネーム・サーバにアクセスするための URL を取得します。

   注意： TIBCO Enterprise Message Services と統合または通信するために JMS 用 Sybase Event Stream Processor アダプタを使用するには、TIBCO または TIBCO の認証されたチャネルから TIBCO Enterprise Message Services の有効なライセンスを取得する必要があります。

   たとえば、Apache Active MQ の場合、これらは、`org.apache.activemq.jndi.ActiveMQInitialContextFactory` と `tcp://localhost:61616` です。

4. JMS アダプタの **jndiContextFactory** プロパティと **jndiURL** プロパティを設定します。
5. JMS 接続ファクトリがバインドされる名前を取得します。
6. アダプタの **connectionFactory** プロパティをこの名前に設定します。
7. 該当する `JNDI` クラスと `JMS factory` クラスが Java クラス・パスにあることを確認します。

JMS CSV インプット・アダプタ

アダプタのタイプ： jms_csv_in。JMS CSV インプット・アダプタは、値の区切られたリストとしてフォーマットされたテキスト・メッセージにサブスクライブし、それらをストリーム・レコードとして書き込みます。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは jms_csv_in です。

`delimiter` をカンマ (,) に、`expectStreamNameOpcode` を true に設定すると、JMS CSV インプット・アダプタは入力ストリームが次のフォーマットであると想定します。

```
aaa,
11,111,1.100000,2008-03-13T08:19:30,111.1111,2008-03-13T08:19:30.12
3,false,FF00FE05FF,
2008-03-13T08:19:30.123456,64000,922.0,337.000000000000000
```

ストリームには、以下のカラムがあります。

- stringCol
- int32Col
- int64Col
- doubleCol
- dateCol

- moneyCol
- timestampCol
- booleanCol
- binaryCol
- bigdatetimeCol
- intervalCol
- money1Col
- money15Col

```
<RecordType name="StreamIn_rec">
    <Column datatype="string" key="true" name="stringCol" />
    <Column datatype="integer" key="false" name="int32Col" />
    <Column datatype="long" key="false" name="int64Col" />
    <Column datatype="float" key="false" name="doubleCol" />
    <Column datatype="date" key="false" name="dateCol" />
    <Column datatype="money" key="false" name="moneyCol" />
    <Column datatype="timestamp" key="false" name="timestampCol" />
    <Column datatype="boolean" key="false" name="booleanCol" />
    <Column datatype="binary" key="false" name="binaryCol" />
    <Column datatype="bigdatetime" key="false"
name="bigdatetimeCol" />
    <Column datatype="interval" key="false" name="intervalCol" />
    <Column datatype="money(1)" key="false" name="money1Col" />
    <Column datatype="money(15)" key="false" name="money15Col" /
>
</RecordType>
```

プロパティ・ラベル	プロパティ ID	タイプ	説明
Delimiter	**delimiter**	`string`	(必須) フィールドの区切り文字。デフォルト値は、カンマ (,) です。
Connection Factory	**connectionFactory**	`string`	(必須) 接続ファクトリのクラス名。 デフォルト値はありません。
JNDI Context Factory	**jndiContextFactory**	`string`	(必須) JNDI コンテキストを初期化するためのコンテキスト・ファクトリ。 デフォルト値はありません。
JNDI URL	**jndiURL**	`string`	(必須) JNDI URL。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Destination Type	**destinationType**	`choice`	(必須) 送信先のタイプ。 有効な値は、次の 2 つです。 • QUEUE • TOPIC デフォルト値は、QUEUE です。
Destination Name	**destinationName**	`string`	(必須) 送信先の名前。 デフォルト値はありません。
Subscription Mode	**subscriptionMode**	`choice`	(オプション) TOPIC のサブスクリプション・モードを指定。 デフォルト値は、NONDURABLE です。 有効な値は、DURABLE と NONDURABLE です。
Client ID	**clientID**	`string`	(オプション) 持続的サブスクリプションを識別する接続のクライアント識別子を指定。 デフォルト値はありません。
Subscription Name	**subscriptionName**	`string`	(オプション) 持続的サブスクリプションを識別する一意の名前を指定。 デフォルト値はありません。
Batch Size	**batchsize**	`uint`	(オプション) 持続的サブスクリプション・モードでコミットするバッチのレコード数を指定する。 デフォルト値は 1 です。
Stream Name Opcode Expected	**expectStreamNameOpcode**	`boolean`	(詳細) true の場合、CSV レコードの先頭の 2 つのフィールドはストリーム名と opcode として解釈される。 デフォルト値は、false です。 **注意：** 空の文字列は有効な値です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Date Format	**dateFormat**	`string`	(詳細) 日付のフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプのフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

既知の制限事項：

- メッセージ・ブローカへの接続が失われても、アダプタは再接続しようとしない。

参照：
- *第 5 章、「保証された配信」* (591 ページ)

JMS CSV アウトプット・アダプタ

アダプタのタイプ： jms_csv_out。 JMS CSV アウトプット・アダプタは、ストリーム・データを、値の区切られたリストとしてフォーマットされたテキスト・メッセージにして JMS のキューまたはトピックにパブリッシュします。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは jms_csv_out です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Delimiter	**delimiter**	`string`	(必須) フィールドの区切り文字。 デフォルト値は、カンマ (,) です。
Connection Factory	**connectionFactory**	`string`	(必須) 接続ファクトリのクラス名。 デフォルト値はありません。
JNDI Context Factory	**jndiContextFactory**	`string`	(必須) JNDI コンテキストを初期化するためのコンテキスト・ファクトリ。 デフォルト値はありません。
JNDI URL	**jndiURL**	`string`	(必須) JNDI URL。 デフォルト値はありません。
Destination Type	**destinationType**	`choice`	(必須) 送信先のタイプ。 有効な値は、次の 2 つです。 • QUEUE • TOPIC デフォルト値は、QUEUE です。
Destination Name	**destinationName**	`string`	(必須) 送信先の名前。 デフォルト値はありません。
Delivery Mode	**deliveryMode**	`choice`	(オプション) 配信モードのタイプ。 有効な値は、次の 2 つです。 • PERSISTENT • NON_PERSISTENT デフォルト値は、PERSISTENT です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。
Column To Message Property Map	**columnPropertyMap**	`string`	(詳細) コンマ区切りの `ColumnName=PropertyName` マッピングのリストで、JMS セレクタ・メカニズムを使用してメッセージ・ブローカ上のメッセージ・フィルタリングを有効にする。 マップされた各カラム名に対して、出力メッセージが、カラム値と等しい値を持つ、対応する JMS プロパティと組み合わされます。 `ColumnName1=PropertyName1,ColumnName2=PropertyName2...` デフォルト値はありません。 **注意：** このプロパティでは、値にスペースは指定できません。
Prepend Stream Name, Opcode	**prependStreamNameOpcode**	`boolean`	(詳細) true の場合、各 CSV レコードの前にストリーム名と opcode が付加される。 デフォルト値はありません。
Date Format	**dateFormat**	`string`	(詳細) 日付のフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプのフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Runs Adapter in GD Mode	**enableGDMode**	`boolean`	(詳細) true に設定すると、アダプタが、保証された配信 (GD) モードで動作し、GD 関連のすべてのパラメータが必須になる。デフォルト値は false です。
Name of Column Holding GD Key	**gdKeyColumn**	`string`	(詳細) GD キーを保持するフレックス演算子のカラム名を指定する。 GD キーは、対象となるストリームに格納される opcode に関係なく、各イベントを一意に識別する、常に増加する値です。 デフォルト値はありません。
Name of Column Holding opcode	**gdOpcodeColumn**	`string`	(詳細) opcode を保持するフレックス演算子のカラム名を指定する。 opcode は、対象のストリームで発生するイベントのオペレーション・コード (たとえば、挿入、更新、または削除) です。 デフォルト値はありません。
Name of Truncate Stream	**gdControlStream**	`string`	(詳細) GD セットアップの制御ウィンドウの名前を指定する。 制御ウィンドウはソース・ストリームで、そのデータがアダプタによって処理済みで、安全に削除できることをフレックス演算子に通知します。 デフォルト値はありません。
Purge After Number of Records	**gdPurgeInternal**	`int`	(詳細) レコード数を指定し、この数に到達すると、フレックス演算子はパージを実行する。デフォルト値は 1000 です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Batch Size to Update Truncate Stream	**gdBatchSize**	`int`	(詳細) レコード数を指定し、この数に到達すると、制御ウィンドウは最新の GD キーで更新される。デフォルト値は 1000 です。

既知の制限事項：

- メッセージ・ブローカへの接続が失われても、アダプタは再接続しようとしない。

参照：

- *第 5 章、「保証された配信」* (591 ページ)

JMS カスタム・インプット・アダプタ

アダプタのタイプ： jms_custom_in。JMS カスタム・インプット・アダプタは、JMS のキューまたはトピックからのカスタムフォーマットの Java オブジェクト・メッセージにサブスクライブし、これらのメッセージをストリーム・レコードとして書き込みます。

カスタム提供の実装によって、このインタフェースのフォーマット変換が実行されます。

```
package com.sybase.esp.adapters;
public interface ExternalToESPConverter {
    public ESPMessage externalToESP(Serializable externalMessage)
throws Exception;
}
```

`externalToESP` メソッドによって返されるオブジェクトがこのインタフェースを実装することを確認します。

```
package com.sybase.esp.adapters;
public interface ESPMessage extends Serializable {
    public String getStreamName();
    public String getOpCode();
    public Map<String, Serializable> getColumnValues();
}
```

`getStreamName`、`getOpCode`、`getColumnValues` のメソッドで返されるオブジェクトは、それぞれ、書き込み先のストリームの名前、opcode、メッセージ・プロパティ・マップ値のカラムとしてのストリーム・レコードとして解釈されます。

ストリーム・カラムの型が、Java クラスに以下のように対応することを確認します。

ストリーム・カラムの型	Java クラス
bigdatetime	java.lang.Double
binary	java.lang.String
boolean	java.lang.Boolean
integer	java.lang.Integer
interval	java.lang.Long
date	java.util.Date
float	java.lang.Double
long	java.lang.Long
money1	java.math.BigDecimal
money2	java.math.BigDecimal
money3	java.math.BigDecimal
money4	java.math.BigDecimal
money5	java.math.BigDecimal
money6	java.math.BigDecimal
money7	java.math.BigDecimal
money8	java.math.BigDecimal
money9	java.math.BigDecimal
money10	java.math.BigDecimal
money11	java.math.BigDecimal
money12	java.math.BigDecimal
money13	java.math.BigDecimal
money14	java.math.BigDecimal
money15	java.math.BigDecimal
string	java.lang.String
timestamp	java.util.Date

ExternalToEFSConverter インタフェースが、`java.lang.String` 型の 1 つの引数を使用するコンストラクタ、または引数のないデフォルト・コンストラクタを提供することを確認します。

注意： ストリーム名に一致しないレコードは、無視されます。 null opcode のレコードは、upsert として解釈されます。キー以外のカラムの値は、存在しなくても、null 値であっても問題ありません。

実装が提供されない場合、デフォルトの実装が代わりに使用されます。この場合、各外部メッセージは DefaultEFSMessage クラスのインスタンスとして解釈され、変換は実行されません。

ExternalToEFSConverter インタフェースの実装のある Java アーカイブが提供され、Event Stream Processor インストール・フォルダの `lib` サブフォルダに配置されていることを確認します。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは jms_custom_in です。

注意： このアダプタは、スキーマ検出をサポートします。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Converter Class Name	**converterClassName**	`string`	(必須) 外部から ESP メッセージへのコンバータの完全修飾されたクラス名。 デフォルト値はありません。
Connection Factory	**connectionFactory**	`string`	(必須) 接続ファクトリのクラス名。 デフォルト値はありません。
JNDI Context Factory	**jndiContextFactory**	`string`	(必須) JNDI コンテキストを初期化するためのコンテキスト・ファクトリ。 デフォルト値はありません。
JNDI URL	**jndiURL**	`string`	(必須) JNDI URL。 デフォルト値はありません。
Destination Type	**destinationType**	`choice`	(必須) 送信先のタイプ。 有効な値は、次の 2 つです。 • QUEUE • TOPIC デフォルト値は、QUEUE です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Destination Name	**destinationName**	`string`	(必須) 送信先の名前。 デフォルト値はありません。
Converter Parameter	**converterParam**	`string`	(オプション) 外部から Event Stream Processor へのメッセージ・コンバータのスタートアップ・パラメータ。 デフォルト値はありません。
Subscription Mode	**subscriptionMode**	`choice`	(オプション) TOPIC のサブスクリプション・モードを指定。 デフォルト値は、NONDURABLE です。 有効な値は、DURABLE と NONDURABLE です。
Client ID	**clientID**	`string`	(オプション) 持続的サブスクリプションを識別する接続のクライアント識別子を指定。 デフォルト値はありません。
Subscription Name	**subscriptionName**	`string`	(オプション) 持続的サブスクリプションを識別する一意の名前を指定。 デフォルト値はありません。
Batch Size	**batchsize**	`uint`	(オプション) 持続的サブスクリプション・モードでコミットするバッチのレコード数を指定する。デフォルト値は 1 です。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。
Date Format	**dateFormat**	`string`	(詳細) 日付のフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプのフォーマット。デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。

既知の制限事項：

- メッセージ・ブローカへの接続が失われても、アダプタは再接続しようとしない。

参照：
- *第 5 章、「保証された配信」* (591 ページ)

JMS カスタム・アウトプット・アダプタ

アダプタのタイプ： jms_custom_out。 JMS カスタム・アウトプット・アダプタは、ストリーム・レコードをカスタムフォーマットの Java オブジェクトとして JMS のキューまたはトピックにパブリッシュします。

カスタム提供の実装によって、このインタフェースのフォーマット変換が実行されます。

```
package com.sybase.esp.adapters;
public interface ESPToExternalConverter {
    public Serializable ESPToExternal(ESPMessage ESPMessage) throws
Exception;
}
```

ストリーム・カラムの型が、Java クラスに以下のように対応することを確認します。

ストリーム・カラムの型	Java クラス
`bigdatetime`	`java.lang.Double`
`binary`	`java.lang.String`
`boolean`	`java.lang.Boolean`
`integer`	`java.lang.Integer`
`interval`	`java.lang.Long`
`date`	`java.util.Date`
`float`	`java.lang.Double`

ストリーム・カラムの型	Java クラス
long	java.lang.Long
money1	java.math.BigDecimal
money2	java.math.BigDecimal
money3	java.math.BigDecimal
money4	java.math.BigDecimal
money5	java.math.BigDecimal
money6	java.math.BigDecimal
money7	java.math.BigDecimal
money8	java.math.BigDecimal
money9	java.math.BigDecimal
money10	java.math.BigDecimal
money11	java.math.BigDecimal
money12	java.math.BigDecimal
money13	java.math.BigDecimal
money14	java.math.BigDecimal
money15	java.math.BigDecimal
string	java.lang.String
timestamp	java.util.Date

ESPToExternalConverter インタフェースが、`java.lang.String` 型の 1 つの引数を使用するコンストラクタ、または引数のないデフォルト・コンストラクタを提供することを確認します。

注意： ESPMessage オブジェクトのストリーム名、opcode、カラム名値のマップは、一部のキー以外のカラムが null の場合でも、有効であることが保証されます。

ExternalToEFSConverter インタフェースの実装のある Java アーカイブが提供され、Event Stream Processor インストール・フォルダの `lib` サブフォルダに配置されていることを確認します。

実装が提供されていない場合、デフォルトの実装が使用され、ESPMessage オブジェクトは変換が実際に実行されずに返されます。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは jms_custom_out です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Converter Class Name	**converterClassName**	`string`	(必須) 外部から Event Stream Processor メッセージへのコンバータの完全修飾されたクラス名。 デフォルト値はありません。
Connection Factory	**connectionFactory**	`string`	(必須) 接続ファクトリのクラス名。 デフォルト値はありません。
JNDI Context Factory	**jndiContextFactory**	`string`	(必須) JNDI コンテキストを初期化するためのコンテキスト・ファクトリ。 デフォルト値はありません。
JNDI URL	**jndiURL**	`string`	(必須) JNDI URL。 デフォルト値はありません。
Destination Type	**destinationType**	`choice`	(必須) 送信先のタイプ。 有効な値は、次の 2 つです。 • QUEUE • TOPIC デフォルト値は、QUEUE です。
Destination Name	**destinationName**	`string`	(必須) 送信先の名前。 デフォルト値はありません。
Delivery Mode	**deliveryMode**	`choice`	(オプション) 配信モードのタイプ。 有効な値は、次の 2 つです。 • PERSISTENT • NON_PERSISTENT デフォルト値は、PERSISTENT です。
Converter Parameter	**converterParam**	`string`	(オプション) 外部から ESP へのメッセージ・コンバータのスタートアップ・パラメータ。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。デフォルト値はありません。
Column To Message Property Map	**columnPropertyMap**	`string`	(詳細) コンマ区切りの `ColumnName=PropertyName` マッピングのリストで、JMS セレクタ・メカニズムを使用してメッセージ・ブローカ側のメッセージ・フィルタリングを有効にする。 マップされた各カラム名に対して、アウトバウンド・メッセージが、カラム値に等しい値を持つ、対応する JMS プロパティと組み合わされます。 `ColumnName1=PropertyName1,ColumnName2=PropertyName2...` デフォルト値はありません。 **注意：** このプロパティでは、値にスペースは指定できません。
Date Format	**dateFormat**	`string`	(詳細) 日付のフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプのフォーマット。デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Runs Adapter in GD Mode	**enableGDMode**	`boolean`	(詳細) true に設定すると、アダプタが、保証された配信 (GD) モードで動作し、GD 関連のすべてのパラメータが必須になる。デフォルト値は false です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Name of Column Holding GD Key	**gdKeyColumn**	`string`	(詳細) GD キーを保持するフレックス演算子のカラム名を指定する。 GD キーは、対象となるストリームに格納される opcode に関係なく、各イベントを一意に識別する、常に増加する値です。 デフォルト値はありません。
Name of Column Holding opcode	**gdOpcodeColumn**	`string`	(詳細) opcode を保持するフレックス演算子のカラム名を指定する。 opcode は、対象のストリームで発生するイベントのオペレーション・コード (たとえば、挿入、更新、または削除) です。 デフォルト値はありません。
Name of Truncate Stream	**gdControlStream**	`string`	(詳細) GD セットアップの制御ウィンドウの名前を指定する。 制御ウィンドウはソース・ストリームで、そのデータがアダプタによって処理済みで、安全に削除できることをフレックス演算子に通知します。 デフォルト値はありません。
Purge After Number of Records	**gdPurgeInternal**	`int`	(詳細) レコード数を指定し、この数に到達すると、フレックス演算子はパージを実行する。デフォルト値は 1000 です。
Batch Size to Update Truncate Stream	**gdBatchSize**	`int`	(詳細) レコード数を指定し、この数に到達すると、制御ウィンドウは最新の GD キーで更新される。デフォルト値は 1000 です。

既知の制限事項：

- メッセージ・ブローカへの接続が失われても、アダプタは再接続しようとしない。

参照：

- *第 5 章、「保証された配信」* (591 ページ)

JMS FIX インプット・アダプタ

アダプタのタイプ： jms_fix_in。JMS FIX インプット・アダプタは、JMS のキューまたはトピックからのメッセージにサブスクライブし、これらのメッセージをストリーム・レコードとして書き込みます。

各ストリームは、特定のタイプの FIX メッセージをホストします。 アダプタは、他のいずれの FIX タイプのメッセージも破棄します。 ほとんどの FIX フィールドはストリーム・カラムと同じ順序で格納されますが、次のフィールドは異なる順序で格納できます。

- BeginString
- BodyLength
- MsgType
- CheckSum

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは jms_fix_in です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
FIX Version	**fixVersion**	`choice`	(必須) FIX のバージョン。 有効な値は次のとおりです。 • 4.2 • 4.3 • 4.4 • 5.0 デフォルト値は 4.2 です。
FIX Message Type	**fixMessageType**	`string`	(必須) FIX メッセージ・タイプ。 デフォルト値はありません。
Connection Factory	**connectionFactory**	`string`	(必須) 接続ファクトリのクラス名。 デフォルト値はありません。
JNDI Context Factory	**jndiContextFactory**	`string`	(必須) JNDI コンテキストを初期化するためのコンテキスト・ファクトリ。 デフォルト値はありません。
JNDI URL	**jndiURL**	`string`	(必須) JNDI URL。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Destination Type	**destinationType**	`choice`	(必須) 送信先のタイプ。 有効な値は次のとおりです。 • QUEUE • TOPIC デフォルト値は QUEUE です。
Destination Name	**destinationName**	`string`	(必須) 送信先の名前。 デフォルト値はありません。
Subscription Mode	**subscriptionMode**	`choice`	(オプション) TOPIC のサブスクリプション・モードを指定。 デフォルト値は、NONDURABLE です。 有効な値は、DURABLE と NONDURABLE です。
Client ID	**clientID**	`string`	(オプション) 持続的サブスクリプションを識別する接続のクライアント識別子を指定。 デフォルト値はありません。
Subscription Name	**subscriptionName**	`string`	(オプション) 持続的サブスクリプションを識別する一意の名前を指定。 デフォルト値はありません。
Batch Size	**batchsize**	`uint`	(オプション) 持続的サブスクリプション・モードでコミットするバッチのレコード数を指定する。デフォルト値は 1 です。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。
Date Format	**dateFormat**	`string`	(詳細) 日付のフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプのフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。

既知の制限事項：

- このアダプタは、完全な FIX エンジンではない。
- FIX バージョン 4.2、4.3、4.4、5.0 のみをサポートする。
- 繰り返しグループと繰り返しコンポーネントはサポートされない。
- メッセージ・ブローカへの接続が失われても、アダプタは再接続しようとしない。
- INSERT opcode のみをサポートする。

参照：
- *第 5 章、「保証された配信」* (591 ページ)

JMS FIX アウトプット・アダプタ

アダプタのタイプ： jms_fix_out。 JMS FIX アウトプット・アダプタは、FIX メッセージを JMS のキューまたはトピックにパブリッシュします。

各ストリームは、特定のタイプの FIX メッセージをホストします。 他のすべてのタイプの FIX メッセージは破棄されます。 以下を除くすべての FIX フィールドは、ストリーム・カラムと同じ順序で格納されます。

- BeginString
- BodyLength
- MsgType
- CheckSum

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは jms_fix_out です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
FIX Version	**fixVersion**	`choice`	(必須) FIX のバージョン。 有効な値は次のとおりです。 • 4.2 • 4.3 • 4.4 • 5.0 デフォルト値は 4.2 です。
FIX Message Type	**fixMessageType**	`string`	(必須) FIX メッセージ・タイプ。
Connection Factory	**connectionFactory**	`string`	(必須) 接続ファクトリのクラス名。
JNDI Context Factory	**jndiContextFactory**	`string`	(必須) JNDI コンテキストを初期化するためのコンテキスト・ファクトリ。
JNDI URL	**jndiURL**	`string`	(必須) JNDI URL。
Destination Type	**destinationType**	`choice`	(必須) 送信先のタイプ。 有効な値は次のとおりです。 • QUEUE • TOPIC デフォルト値は QUEUE です。
Destination Name	**destinationName**	`string`	(必須) 送信先の名前。
Delivery Mode	**deliveryMode**	`choice`	(オプション) 配信モードのタイプ。 有効な値は次のとおりです。 • PERSISTENT • NON_PERSISTENT デフォルト値は PERSISTENT です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。デフォルト値はありません。
Column To Message Property Map	**columnPropertyMap**	`string`	(詳細) カンマ区切りの MyColumn=MyMessageProperty 対応リスト。
Date Format	**dateFormat**	`string`	(詳細) 日付のフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプのフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Runs Adapter in GD Mode	**enableGDMode**	`boolean`	(詳細) true に設定すると、アダプタが、保証された配信 (GD) モードで動作し、GD 関連のすべてのパラメータが必須になる。デフォルト値は false です。
Name of Column Holding GD Key	**gdKeyColumn**	`string`	(詳細) GD キーを保持するフレックス演算子のカラム名を指定する。 GD キーは、対象となるストリームに格納される opcode に関係なく、各イベントを一意に識別する、常に増加する値です。 デフォルト値はありません。
Name of Column Holding opcode	**gdOpcodeColumn**	`string`	(詳細) opcode を保持するフレックス演算子のカラム名を指定する。 opcode は、対象のストリームで発生するイベントのオペレーション・コード (たとえば、挿入、更新、または削除) です。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Name of Truncate Stream	**gdControlStream**	`string`	(詳細) GD セットアップの制御ウィンドウの名前を指定する。 制御ウィンドウはソース・ストリームで、そのデータがアダプタによって処理済みで、安全に削除できることをフレックス演算子に通知します。 デフォルト値はありません。
Purge After Number of Records	**gdPurgeInternal**	`int`	(詳細) レコード数を指定し、この数に到達すると、フレックス演算子はパージを実行する。デフォルト値は 1000 です。
Batch Size to Update Truncate Stream	**gdBatchSize**	`int`	(詳細) レコード数を指定し、この数に到達すると、制御ウィンドウは最新の GD キーで更新される。デフォルト値は 1000 です。

既知の制限事項：

- このアダプタは、完全な FIX エンジンではない。
- FIX バージョン 4.2、4.3、4.4、5.0 のみをサポートする。
- 繰り返しグループと繰り返しコンポーネントはサポートされない。
- スキーマ検出はサポートされない。
- メッセージ・ブローカへの接続が失われても、アダプタは再接続しようとしない。
- INSERT opcode のみをサポートする。

参照：
- *第 5 章、「保証された配信」* (591 ページ)

JMS Object Array インプット・アダプタ

アダプタのタイプ： jms_objarray_in。JMS Object Array インプット・アダプタは、JMS のキューまたはトピックからの Java オブジェクトの配列としてフォーマットされたメッセージにサブスクライブし、これらのメッセージをストリーム・レコードとして書き込みます。

注意： 配列内に null 要素があると、対応するカラムに対して null 値が生成されます。

ストリーム・カラムの型が、Java クラスに以下のように対応することを確認します。

ストリーム・カラムの型	Java クラス
bigdatetime	java.lang.Double
binary	java.lang.String
boolean	java.lang.Boolean
integer	java.lang.Integer
interval	java.lang.Long
date	java.util.Date
float	java.lang.Double
long	java.lang.Long
money1	java.math.BigDecimal
money2	java.math.BigDecimal
money3	java.math.BigDecimal
money4	java.math.BigDecimal
money5	java.math.BigDecimal
money6	java.math.BigDecimal
money7	java.math.BigDecimal
money8	java.math.BigDecimal
money9	java.math.BigDecimal
money10	java.math.BigDecimal
money11	java.math.BigDecimal
money12	java.math.BigDecimal
money13	java.math.BigDecimal
money14	java.math.BigDecimal
money15	java.math.BigDecimal
string	java.lang.String
timestamp	java.util.Date

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは jms_objarray_in です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Connection Factory	**connectionFactory**	`string`	(必須) 接続ファクトリのクラス名。 デフォルト値はありません。
JNDI Context Factory	**jndiContextFactory**	`string`	(必須) JNDI コンテキストを初期化するためのコンテキスト・ファクトリ。 デフォルト値はありません。
JNDI URL	**jndiURL**	`string`	(必須) JNDI URL。 デフォルト値はありません。
Destination Type	**destinationType**	`choice`	(必須) 送信先のタイプ。 有効な値は次のとおりです。 • QUEUE • TOPIC デフォルト値は QUEUE です。
Destination Name	**destinationName**	`string`	(必須) 送信先の名前。 デフォルト値はありません。
Subscription Mode	**subscriptionMode**	`choice`	(オプション) TOPIC のサブスクリプション・モードを指定。 デフォルト値は、NONDURABLE です。 有効な値は、DURABLE と NONDURABLE です。
Client ID	**clientID**	`string`	(オプション) 持続的サブスクリプションを識別する接続のクライアント識別子を指定。 デフォルト値はありません。
Subscription Name	**subscriptionName**	`string`	(オプション) 持続的サブスクリプションを識別する一意の名前を指定。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Batch Size	**batchsize**	`uint`	(オプション) 持続的サブスクリプション・モードでコミットするバッチのレコード数を指定する。 デフォルト値は 1 です。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。
Stream Name Opcode Expected	**expectStreamNameOpcode**	`boolean`	(詳細) true の場合、CSV レコードの先頭の 2 つのフィールドはストリーム名と opcode として解釈される。デフォルト値は false です。
Date Format	**dateFormat**	`string`	(詳細) 日付のフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプのフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。

既知の制限事項：

- メッセージ・ブローカへの接続が失われても、アダプタは再接続しようとしない。

参照：

- *第 5 章、「保証された配信」* (591 ページ)

JMS Object Array アウトプット・アダプタ

アダプタのタイプ： jms_objarray_out。 JMS Object Array アウトプット・アダプタは、ストリーム・データを Java オブジェクトの配列として JMS のキューまたはトピックにパブリッシュします。

注意： カラムが null 値の場合、対応する配列に対して null 要素が生成されます。

ストリーム・カラムの型が、Java クラスに以下のように対応することを確認します。

ストリーム・カラムの型	Java クラス
bigdatetime	java.lang.Double
binary	java.lang.String
boolean	java.lang.Boolean
integer	java.lang.Integer
interval	java.lang.Long
date	java.util.Date
float	java.lang.Double
long	java.lang.Long
money1	java.math.BigDecimal
money2	java.math.BigDecimal
money3	java.math.BigDecimal
money4	java.math.BigDecimal
money5	java.math.BigDecimal
money6	java.math.BigDecimal
money7	java.math.BigDecimal
money8	java.math.BigDecimal
money9	java.math.BigDecimal
money10	java.math.BigDecimal
money11	java.math.BigDecimal

ストリーム・カラムの型	Java クラス
money12	java.math.BigDecimal
money13	java.math.BigDecimal
money14	java.math.BigDecimal
money15	java.math.BigDecimal
string	java.lang.String
timestamp	java.util.Date

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは jms_objarray_out です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Connection Factory	**connectionFactory**	string	(必須) 接続ファクトリのクラス名。 デフォルト値はありません。
JNDI Context Factory	**jndiContextFactory**	string	(必須) JNDI コンテキストを初期化するためのコンテキスト・ファクトリ。 デフォルト値はありません。
JNDI URL	**jndiURL**	string	(必須) JNDI URL。 デフォルト値はありません。
Destination Type	**destinationType**	choice	(必須) 送信先のタイプ。 有効な値は次のとおりです。 • QUEUE • TOPIC デフォルト値は QUEUE です。
Destination Name	**destinationName**	string	(必須) 送信先の名前。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Delivery Mode	**deliveryMode**	`choice`	(オプション) 配信モードのタイプ。 有効な値は次のとおりです。 • PERSISTENT • NON_PERSISTENT デフォルト値は PERSISTENT です。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。
Column To Message Property Map	**columnPropertyMap**	`string`	(詳細) コンマ区切りの `ColumnName=PropertyName` マッピングのリストで、JMS セレクタ・メカニズムを使用してメッセージ・ブローカ側のメッセージ・フィルタリングを有効にする。 マップされた各カラム名に対して、アウトバウンド・メッセージが、カラム値に等しい値を持つ、対応する JMS プロパティと組み合わされます。 `ColumnName1=PropertyName1,ColumnName2=PropertyName2...` デフォルト値はありません。 **注意：** このプロパティでは、値にスペースは指定できません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Prepend Stream Name, Opcode	**prependStreamNameOpcode**	`boolean`	(詳細) true の場合、各 CSV レコードの前にストリーム名と opcode が付加される。 デフォルト値はありません。
Date Format	**dateFormat**	`string`	(詳細) 日付のフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプのフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Runs Adapter in GD Mode	**enableGDMode**	`boolean`	(詳細) true に設定すると、アダプタが、保証された配信 (GD) モードで動作し、GD 関連のすべてのパラメータが必須になる。デフォルト値は false です。
Name of Column Holding GD Key	**gdKeyColumn**	`string`	(詳細) GD キーを保持するフレックス演算子のカラム名を指定する。 GD キーは、対象となるストリームに格納される opcode に関係なく、各イベントを一意に識別する、常に増加する値です。 デフォルト値はありません。
Name of Column Holding opcode	**gdOpcodeColumn**	`string`	(詳細) opcode を保持するフレックス演算子のカラム名を指定する。 opcode は、対象のストリームで発生するイベントのオペレーション・コード (たとえば、挿入、更新、または削除) です。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Name of Truncate Stream	**gdControlStream**	`string`	(詳細) GD セットアップの制御ウィンドウの名前を指定する。 制御ウィンドウはソース・ストリームで、そのデータがアダプタによって処理済みで、安全に削除できることをフレックス演算子に通知します。 デフォルト値はありません。
Purge After Number of Records	**gdPurgeInternal**	`int`	(詳細) レコード数を指定し、この数に到達すると、フレックス演算子はパージを実行する。デフォルト値は 1000 です。
Batch Size to Update Truncate Stream	**gdBatchSize**	`int`	(詳細) レコード数を指定し、この数に到達すると、制御ウィンドウは最新の GD キーで更新される。デフォルト値は 1000 です。

既知の制限事項：

- メッセージ・ブローカへの接続が失われても、アダプタは再接続しようとしない。

参照：
- *第 5 章、「保証された配信」* (591 ページ)

JMS XML インプット・アダプタ

アダプタのタイプ： jms_xml_in。JMS XML インプット・アダプタは、JMS のキューまたはトピックからの XML フォーマットのテキスト・メッセージにサブスクライブし、これらのメッセージをストリーム・レコードとして書き込みます。

各メッセージが XML 要素で構成されていることを確認します。 選択した場合、要素名は、ストリーム名に対応します。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは jms_xml_in です。

次は、データ・ファイルのレコード・フォーマット例です。

```
< StreamOut ESP_OPS="u" stringCol="aaa" int32Col="22" int64Col="222"
doubleCol="2.200000" dateCol="2008-03-13T08:19:30"
moneyCol="222.2222" timestampCol="2008-03-13T08:19:30.123"
```

```
booleanCol="true" binaryCol="FF00FE05FF"
bigdatetimeCol="2008-03-13T08:19:30.123456" intervalCol="64000"
money1Col="922.0" money15Col="337.000000000000000" />
```

ストリームには、以下のカラムがあります。

- stringCol
- int32Col
- int64Col
- doubleCol
- dateCol
- moneyCol
- timestampCol
- booleanCol
- binaryCol
- bigdatetimeCol
- intervalCol
- money1Col
- money15Col

```
<RecordType name="StreamIn_rec">
    <Column datatype="string" key="true" name="stringCol" />
    <Column datatype="integer" key="false" name="int32Col" />
    <Column datatype="long" key="false" name="int64Col" />
    <Column datatype="float" key="false" name="doubleCol" />
    <Column datatype="date" key="false" name="dateCol" />
    <Column datatype="money" key="false" name="moneyCol" />
    <Column datatype="timestamp" key="false" name="timestampCol" />
    <Column datatype="boolean" key="false" name="booleanCol" />
    <Column datatype="binary" key="false" name="binaryCol" />
    <Column datatype="bigdatetime" key="false"
name="bigdatetimeCol" />
    <Column datatype="interval" key="false" name="intervalCol" />
    <Column datatype="money(1)" key="false" name="money1Col" />
    <Column datatype="money(15)" key="false" name="money15Col" /
>
</RecordType>
```

ESP-OPS 属性はオプションです。 省略すると、メッセージは upsert として解釈されます。 その他の属性が対応するストリームのカラムと同じ名前で、null 値のカラムが省略されていることを確認します。 このアダプタは、スキーマ検出をサポートします。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Connection Factory	**connectionFactory**	`string`	(必須) 接続ファクトリのクラス名。 デフォルト値はありません。
JNDI Context Factory	**jndiContextFactory**	`string`	(必須) JNDI コンテキストを初期化するためのコンテキスト・ファクトリ。 デフォルト値はありません。
JNDI URL	**jndiURL**	`string`	(必須) JNDI URL。 デフォルト値はありません。
Destination Type	**destinationType**	`choice`	(必須) 送信先のタイプ。 有効な値は次のとおりです。 • QUEUE • TOPIC デフォルト値は QUEUE です。
Destination Name	**destinationName**	`string`	(必須) 送信先の名前。 デフォルト値はありません。
Subscription Mode	**subscriptionMode**	`choice`	(オプション) TOPIC のサブスクリプション・モードを指定。 デフォルト値は、NONDURABLE です。 有効な値は、DURABLE と NONDURABLE です。
Client ID	**clientID**	`string`	(オプション) 持続的サブスクリプションを識別する接続のクライアント識別子を指定。 デフォルト値はありません。
Subscription Name	**subscriptionName**	`string`	(オプション) 持続的サブスクリプションを識別する一意の名前を指定。 デフォルト値はありません。
Batch Size	**batchsize**	`uint`	(オプション) 持続的サブスクリプション・モードでコミットするバッチのレコード数を指定する。デフォルト値は 1 です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。
Match Stream Name	**matchStreamName**	`boolean`	(詳細) XML 要素名がソース・ストリーム名に一致しない場合、メッセージを無視。デフォルト値は false です。
Date Format	**dateFormat**	`string`	(詳細) 日付のフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプのフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。

既知の制限事項：

- メッセージ・ブローカへの接続が失われても、アダプタは再接続しようとしない。

参照：
- *第 5 章、「保証された配信」* (591 ページ)

JMS XML アウトプット・アダプタ

アダプタのタイプ： jms_xml_out。 JMS XML アウトプット・アダプタは、ストリーム・データを XML フォーマットのテキスト・メッセージとして JMS のキューまたはトピックにパブリッシュします。

各メッセージが、ストリーム名と同じ名前の XML 要素で構成されていることを確認します。

最初の属性は、Event Stream Processor の opcode です。その他の属性の名前は、対応するストリームのカラムの名前と同じです。 null 値のカラムはすべて省略されていることを確認します。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは jms_xml_out です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Connection Factory	**connectionFactory**	`string`	(必須) 接続ファクトリのクラス名。 デフォルト値はありません。
JNDI Context Factory	**jndiContextFactory**	`string`	(必須) JNDI コンテキストを初期化するためのコンテキスト・ファクトリ。 デフォルト値はありません。
JNDI URL	**jndiURL**	`string`	(必須) JNDI URL。 デフォルト値はありません。
Destination Type	**destinationType**	`choice`	(必須) 送信先のタイプ。 有効な値は次のとおりです。 • QUEUE • TOPIC デフォルト値は QUEUE です。
Destination Name	**destinationName**	`string`	(必須) 送信先の名前。 デフォルト値はありません。
Delivery Mode	**deliveryMode**	`choice`	(オプション) 配信モードのタイプ。 有効な値は次のとおりです。 • PERSISTENT • NON_PERSISTENT デフォルト値は PERSISTENT です。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Column To Message Property Map	**columnPropertyMap**	`string`	(詳細) コンマ区切りの `ColumnName=PropertyName` マッピングのリストで、JMS セレクタ・メカニズムを使用してメッセージ・ブローカ側のメッセージ・フィルタリングを有効にする。 マップされた各カラム名に対して、出力メッセージが、カラム値と等しい値を持つ、対応する JMS プロパティと組み合わされます。 `ColumnName1=PropertyName1,ColumnName2=PropertyName2...` デフォルト値はありません。 **注意：** このプロパティでは、値にスペースは指定できません。
Date Format	**dateFormat**	`string`	(詳細) 日付のフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプのフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Runs Adapter in GD Mode	**enableGDMode**	`boolean`	(詳細) true に設定すると、アダプタが、保証された配信 (GD) モードで動作し、GD 関連のすべてのパラメータが必須になる。 デフォルト値は false です。
Name of Column Holding GD Key	**gdKeyColumn**	`string`	(詳細) GD キーを保持するフレックス演算子のカラム名を指定する。 GD キーは、対象となるストリームに格納される opcode に関係なく、各イベントを一意に識別する、常に増加する値です。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Name of Column Holding opcode	**gdOpcodeColumn**	`string`	(詳細) opcode を保持するフレックス演算子のカラム名を指定する。 opcode は、対象のストリームで発生するイベントのオペレーション・コード (たとえば、挿入、更新、または削除) です。 デフォルト値はありません。
Name of Truncate Stream	**gdControlStream**	`string`	(詳細) GD セットアップの制御ウィンドウの名前を指定する。 制御ウィンドウはソース・ストリームで、そのデータがアダプタによって処理済みで、安全に削除できることをフレックス演算子に通知します。 デフォルト値はありません。
Purge After Number of Records	**gdPurgeInternal**	`int`	(詳細) レコード数を指定し、この数に到達すると、フレックス演算子はパージを実行する。デフォルト値は 1000 です。
Batch Size to Update Truncate Stream	**gdBatchSize**	`int`	(詳細) レコード数を指定し、この数に到達すると、制御ウィンドウは最新の GD キーで更新される。デフォルト値は 1000 です。

既知の制限事項：

- メッセージ・ブローカへの接続が失われても、アダプタは再接続しようとしない。

参照：
- *第 5 章、「保証された配信」* (591 ページ)

ランダム・タプル生成インプット・アダプタ

アダプタのタイプ： randomtuplegen_in。ランダム・タプル生成インプット・アダプタは、指定されたスキーマに従ってランダム・タプルを生成し、ローをストリームに送信します。

このアダプタは、主に Event Stream Processor のプロトタイプ構築と基本テストに使用されます。 スキーマと設定ファイルの両方を編集できます。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは randomtuplegen_in です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Rate	**Rate**	`uint`	(オプション) 1 秒あたりに生成されたロー数。 1 ～ 1,000,000 までの間。 **Rate** が負の場合、アダプタは停止し、致命的なエラー・メッセージをサーバに返します。 **Rate** が最大値より大きい場合、最大値にリセットされ、警告メッセージがサーバにレポートされます。 **Rate** プロパティがブランクの場合、デフォルト値にリセットされ、メッセージがレポートされます。 デフォルト値は 100 です。
Row Count	**RowCount**	`uint`	(オプション) 生成されたローの数を指定する。 0 ～ 2,000,000,000 までの間。 **RowCount** が負の場合、アダプタは停止し、致命的なエラー・メッセージをサーバに返します。 **RowCount** が最大値より大きい場合、最大値にリセットされ、警告メッセージがサーバにレポートされます。 **RowCount** プロパティがブランクの場合、デフォルト値にリセットされ、情報メッセージがサーバにレポートされます。 デフォルト値は 0 で、ローの数が無制限であることを意味します。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Timestamp Base	**TimestampBase**	`string`	(オプション) メッセージ・タイムスタンプの初期時間。 **TimestampBase** のサポートされるフォーマットは、%Y-%m-%dT%H:%M:%S です。 **TimestampBase** がブランクの場合、デフォルト値にリセットされ、すぐにアダプタが開始します。 同様に、**TimestampBase** の値が現在の時刻よりも前である場合、すぐにアダプタが開始します。 **TimestampBase** の値が現在の時刻より後の場合、その時刻までアダプタはスリープします。 **TimestampBase** が無効なタイムスタンプ・フォーマットである場合、アダプタは停止し、致命的なエラー・メッセージをサーバに返します。 デフォルト値は現在の時刻です。
Date Format	**DateFormat**	`string`	(オプション) 日付値を解析するためのフォーマット文字列。 デフォルト値は、`%Y-%m-%d %H:%M:%S` です。
Timestamp Format	**TimestampFormat**	`string`	(オプション) 日付値を解析するためのフォーマット文字列。 デフォルト値は、`%Y-%m-%d %H:%M:%S` です。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

このアダプタが生成するデータは、各データ型の有効な値の範囲にわたって均等に分配されません。 次の表に、各データ型の生成された値の範囲を示します。

データ型	値の範囲
boolean	true/false
integer	0 .. 99
long	0 .. 99
float	0.0 .. 10.0 未満 (10.0 にはなりません)
interval	0 .. 9
timestamp	ミリ秒単位の現在時刻
string	a 〜 z、A 〜 Z、0 〜 9 の範囲の 2 文字
binary	2 バイトでそれぞれが 0 〜 255 の範囲
money	0.0000 〜 3.2767
money1	0.0 〜 3276.7
money2	0.00 〜 327.67
money3	0.000 〜 32.767
money4	0.0000 〜 3.2767
money5	0.00000 〜 0.32767
money6	0.000000 〜 0.032767
money7	0.0000000 〜 0.0032767
money8	0.00000000 〜 0.00032767
money9	0.000000000 〜 0.000032767
money10	0.0000000000 〜 0.0000032767
money11	0.00000000000 〜 0.00000032767
money12	0.000000000000 〜 0.000000032767
money13	0.0000000000000 〜 0.0000000032767
money14	0.00000000000000 〜 0.00000000032767
money15	0.000000000000000 〜 0.000000000032767
bigdatetime	マイクロ秒単位の現在時刻

データ型	値の範囲
`date`	秒単位の現在時刻

注意： 値は、必ずしも、これらの範囲内で均等に分配されません。

ソケット FIX インプット・アダプタ

アダプタのタイプ： fixsocket_in。ソケット FIX インプット・アダプタは、TCP サーバのソケットから FIX メッセージを読み込み、ストリーム・レコードとして書き込みます。

各ストリームは、特定のタイプの FIX メッセージをホストします。 アダプタの他のすべてのタイプの FIX メッセージを廃棄し、ストリーム・カラムと同じ順序で FIX フィールドを格納します。 次のフィールドは、このルールの例外です。

- BeginString
- BodyLength
- MsgType
- CheckSum

ストリーム・カラムの名前が、FIX プロトコルの仕様に対応していることを確認します。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは fixsocket_in です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
FIX Version	**fixVersion**	`choice`	(必須) FIX プロトコルのバージョン。 デフォルト値は 4.2 です。
FIX Message Type	**fixMessageType**	`string`	(必須) ストリームによってホストされるメッセージのタイプ。 デフォルト値はありません。
Source Host	**fixHost**	`string`	(必須) FIX メッセージのソース・サーバの名前または IP アドレス。 デフォルト値は、localhost です。
Source Port	**fixPort**	`uint`	(必須) FIX メッセージが利用できるポート。 デフォルト値は 12345 です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Reconnect Interval	**reconnectInterval**	`uint`	(必須) 秒単位の再接続間隔。 ゼロの場合、再接続は試行されません。デフォルト値は 10 です。
Maximum Reconnect Attempts	**maxReconnectAttempts**	`uint`	(必須) 再接続の最大試行回数。 デフォルト値はゼロです。
Date Format	**dateFormat**	`string`	(詳細) 日付のフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプのフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

既知の制限事項：

- このアダプタは、完全な FIX エンジンではない。
- FIX のバージョン 4.2、4.3、4.4、5.0 のみをサポートする。
- グループとコンポーネントの繰り返しはサポートされない。
- INSERT opcode のみをサポートする。

参照：

- *FIX アダプタ* (147 ページ)

ソケット FIX インプット・アダプタのデータ型のマッピング

Event Stream Processor のデータ型は、FIX データ型にマップされます。

Event Stream Processor のデータ型	QuickFix のデータ型
integer	boolean
string	byte[]
string	char
string	string
date	date
float	float
integer	integer
date または timestamp	UTCDateOnly
date または timestamp	UTCTimeOnly
date または timestamp	UTCTimeStamp

ソケット FIX アウトプット・アダプタ

アダプタのタイプ： fixsocket_out。 ソケット FIX アウトプット・アダプタは、ストリーム・データを FIX メッセージとして TCP サーバのソケットに書き込みます。

各ストリームは、特定のタイプの FIX メッセージをホストします。 アダプタは、改行文字を付加せずに、メッセージを隣接させて送信します。 次の FIX フィールドが生成されます。

- BeginString
- BodyLength
- MsgType
- CheckSum

残りのフィールドが適切な順序でストリーム・カラムに格納され、ストリーム・カラムの名前が、FIX プロトコルの仕様に対応していることを確認します。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは fixsocket_out です。

プロパティ・ラベル	パラメータ ID	タイプ	説明
FIX Version	**fixVersion**	`choice`	(必須) FIX プロトコルのバージョン。デフォルト値は 4.2 です。
FIX Message Type	**fixMessageType**	`string`	(必須) ストリームによってホストされるメッセージのタイプ。 デフォルト値はありません。
Destination Host	**fixHost**	`string`	(必須) FIX メッセージの送信先サーバの名前または IP アドレス。 デフォルト値は、localhost です。
Destination Port	**fixPort**	`uint`	(必須) 送信先サーバのソケットが FIX メッセージを受信するポート。デフォルト値は 12346 です。
Date Format	**dateFormat**	`string`	(詳細) 日付のフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプのフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

既知の制限事項：

- このアダプタは、完全な FIX エンジンではない。
- FIX のバージョン 4.2、4.3、4.4、5.0 のみをサポートする。
- グループとコンポーネントの繰り返しはサポートされない。
- FIX サーバへの接続が失われても再接続しない。
- INSERT opcode のみをサポートする。

参照：

- *FIX アダプタ* (147 ページ)

ソケット FIX アウトプット・アダプタのデータ型のマッピング

Event Stream Processor のデータ型は、FIX データ型にマップされます。

Event Stream Processor のデータ型	QuickFix のデータ型
`integer`	`boolean`
`string`	`byte[]`
`string`	`char`
`string`	`string`
`date`	`date`
`float`	`float`
`integer`	`integer`
`date` または `timestamp`	`UTCDateOnly`
`date` または `timestamp`	`UTCTimeOnly`
`date` または `timestamp`	`UTCTimeStamp`

ソケット (クライアント側) CSV インプット・アダプタ

アダプタのタイプ： dsv_sockout_in。ソケット (クライアント側) CSV インプット・アダプタは、送信ネットワーク・アダプタから区切られたフォーマットでデータを受信します。

アダプタは外部データソースへの接続を開始し、外部プログラムがデータを送信します。 このデータはヘッダを必要としません (**esp_convert** で受け入れられます)。 ファイルにヘッダがある場合、そのヘッダはフィールド名を指定します。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは dsv_sockout_in です。

次は、データ・ファイルのレコード・フォーマット例です。

```
1. hasHeader=true
delimiter=,
expectStreamNameOpcode=false

Ts,ItemID,Price,Quantity,WarehouseZipCode,DeliveryZipCode
2004/06/17 10:00:00.000000,SKU1276532,50.00,1,10012,94086
2004/06/17 10:00:05.000000,SKU6723143,23.00,2,10012,94043
```

```
2. expectStreamNameOpcode=true
delimiter=,

Trades_in,i,2004/06/17
10:00:00.000000,SKU1276532,50.00,1,10012,94086
Trades_in,i,2004/06/17
10:00:05.000000,SKU6723143,23.00,2,10012,94043

3. expectStreamNameOpcode=false
timestampFormat=%Y/%m/%d %H:%M:%S
delimiter=,

2004/06/17 10:00:00.000000,SKU1276532,50.00,1,10012,94086
2004/06/17 10:00:05.000000,SKU6723143,23.00,2,10012,94043
```

プロパティ・ラベル	プロパティ ID	タイプ	説明
Server	**host**	`string`	(必須) サーバ・ホスト名。 デフォルト値は、localhost です。
Port	**port**	`int`	(必須) サーバ・ポート。 **port** が -1 に設定された場合、アダプタはエフェメラルポート・ファイルから読み込みます。デフォルト値は 12345 です。
Stream name, opcode expected	**expectStreamNameOpcode**	`boolean`	(オプション) true の場合、最初の 2 つのフィールドはそれぞれ、ストリーム名と opcode と解釈される。 アダプタはストリーム名に一致しないメッセージを破棄します。デフォルト値は false です。
Field Count	**fieldCount**	`uint`	(オプション) ソース・ストリームと異なる場合、CSV ファイルのフィールドの数をカウントする。デフォルト値は 0 です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Delimiter	**delimiter**	`string`	(詳細) カラムを区切るために使用する記号。 デフォルト値は、カンマ (,) です。
Has Header	**hasHeader**	`boolean`	(詳細) 最初の行がフィールドの説明を含むかどうかを決定する。デフォルト値は false です。
Ephemeral Port File	**epFile**	`filename`	(詳細) **Port** が -1 の場合、サーバのポート番号を格納するファイル。デフォルト値はありません。
Retry Period	**retryperiod**	`uint`	(詳細) 送信接続を再確立しようとする間隔。 秒単位。デフォルト値は 1 です。
Enter Initial State	**initial**	`choice`	(詳細) アダプタがいつ初期ロード状態に入るかを指定する。 デフォルト値は never です。
Convert to Safe Opcodes	**safeOps**	`boolean`	(詳細) opcode の INSERT と UPDATE を UPSERT に変換する。 DELETE を SAFEDELETE に変換します。デフォルト値は false です。
Skip Deletes	**skipDels**	`boolean`	(詳細) DELETE または SAFEDELETE の opcode を含むローをスキップする。デフォルト値は false です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプ値を解析するためのフォーマット文字列。 デフォルト値は、%Y-%m-%dT%H:%M:%S です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Date Format	**dateFormat**	`string`	(詳細) 日付値を解析するためのフォーマット文字列。 デフォルト値は、%Y-%m-%dT%H:%M:%S です。
Block Size	**blockSize**	`int`	(詳細) 擬似トランザクションになる、ブロックあたりのレコード数を決定する。デフォルト値は 1 です。
Field Mapping	**permutation**	`permutation`	(詳細) プラットフォーム固有のフィールドと外部フィールドとの間のマッピング。 デフォルト値はありません。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

既知の制限事項：

- アダプタはファイル・ローのストリーム名を無視する。
- すべてのデータは、同じストリームに送信される。

ソケット (クライアント側) CSV アウトプット・アダプタ

アダプタのタイプ： dsv_sockout_out。 ソケット (クライアント側) CSV アウトプット・アダプタは、送信ネットワークに区切られたフォーマットでデータを送信します。

ソケット (クライアント側) CSV アウトプット・アダプタは外部データソースへの接続を開始し、データを送信します。 このデータはヘッダを必要としません

(**esp_convert** で受け入れられます)。 ファイルにヘッダが含まれる場合、そのヘッダはフィールド名を指定します。 接続が切断されると、アダプタは接続をリトライします。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは dsv_sockout_out です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Server	**host**	`string`	(必須) サーバ・ホスト名。デフォルト値は、localhost です。
Port	**port**	`int`	(必須) サーバ・ポート。**port** が -1 に設定された場合、アダプタはエフェメラルポート・ファイルから読み込みます。デフォルト値は 12345 です。
Prepend stream name, opcode	**prependStreamNameOpcode**	`boolean`	(オプション) true の場合、最初の 2 つのフィールドはそれぞれ、ストリーム名と opcode と解釈される。 アダプタはストリーム名に一致しないメッセージを破棄します。デフォルト値は false です。
Delimiter	**delimiter**	`string`	(詳細) カラムを区切るために使用する記号。 デフォルト値は、カンマ (,) です。
Has Header	**hasHeader**	`boolean`	(詳細) 最初の行がフィールドの説明を含むかどうかを決定する。デフォルト値は false です。
Ephemeral Port File	**epFile**	`filename`	(詳細) **port** が -1 の場合、サーバのポート番号を格納するファイル。デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Retry Period, s	**retryperiod**	`uint`	(詳細) 送信接続を再確立しようとする間隔 (秒単位)。デフォルト値は 1 です。
Include Base Content	**outputBase**	`boolean`	(オプション) ストリームの更新内容だけではなく、初期内容も記録して開始する。デフォルト値は false です。
Only Base Content	**onlyBase**	`boolean`	(詳細) 1 回だけ、ストリームの初期内容を送信する。デフォルト値は false です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプ値を解析するためのフォーマット文字列。 デフォルト値は、%Y-%m-%dT%H:%M:%S です。
Date Format	**dateFormat**	`string`	(詳細) 日付値を解析するためのフォーマット文字列。デフォルト値は、%Y-%m-%dT%H:%M:%S です。
Field Mapping	**permutation**	`permutation`	(詳細) プラットフォーム固有のフィールドと外部フィールドとの間のマッピング。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

ソケット (クライアント側) XML インプット・アダプタ

アダプタのタイプ： xml_sockout_in。ソケット (クライアント側) XML インプット・アダプタは、送信ネットワーク・アダプタから Event Stream Processor フォーマットでデータを受信します。

アダプタは、入力アダプタにデータを送信できる送信ネットワーク・アダプタへの接続を開始します。 データにヘッダを設定しないこともできますし、ヘッダにフィールド名を指定しないこともできます。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは xml_sockout_in です。

次は、データ・ファイルのレコード・フォーマット例です。

```
<Trades Id="0" Symbol="EBAY" TradeTime="2000-05-04T12:00:00"
Price="140.0" Shares="50" />
<Trades Id="1" Symbol="EBAY" TradeTime="2000-05-04T12:00:01"
Price="150.0" Shares="500" />
```

プロパティ・ラベル	プロパティ ID	タイプ	説明
Server	**host**	`string`	(必須) サーバ・ホスト名。 デフォルト値は、localhost です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Port	**port**	`int`	(必須) サーバ・ポート。 **port** が -1 に設定された場合、アダプタはエフェメラルポート・ファイルから読み込みます。デフォルト値は 12345 です。
Match Stream Name	**matchStreamName**	`boolean`	(オプション) XML 要素名がソース・ストリーム名に一致しない場合、メッセージを無視する。デフォルト値は false です。
Ephemeral Port File	**epFile**	`filename`	(詳細) **port** が -1 に設定された場合に、サーバのポート番号を格納するファイル。デフォルト値は false です。
Retry period (秒)	**retryperiod**	`uint`	(詳細) 送信接続を再確立しようとする間隔 (秒単位) を指定する。デフォルト値は 1 です。
Enter Initial State	**initial**	`choice`	(詳細) アダプタがいつ初期ロード状態に入るかを指定する。デフォルト値は never です。
Convert to Safe Opcodes	**safeOps**	`boolean`	(詳細) opcode の INSERT と UPDATE を UPSERT に変換し、DELETE を SAFEDELETE に変換する。デフォルト値は false です。
Skip Deletes	**skipDels**	`boolean`	(詳細) DELETE または SAFEDELETE の opcode を含むローをスキップする。デフォルト値は false です。
Date Format	**dateFormat**	`string`	(詳細) 日付値を解析するためのフォーマット文字列。デフォルト値は、%Y-%m-%dT%H:%M:%S です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプ値を解析するためのフォーマット文字列。 デフォルト値は、%Y-%m-%dT%H:%M:%S です。
Block Size	**blockSize**	`int`	(詳細) 擬似トランザクションになる、ブロックあたりのレコード数。 デフォルト値は 1 です。
Field Mapping	**permutation**	`permutation`	(詳細) プラットフォーム固有のフィールドと外部フィールドとの間のマッピング。 デフォルト値はありません。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

既知の制限事項：

- アダプタはファイル・ローのストリーム名を無視する。
- すべてのデータは、同じストリームに送信される。

ソケット (クライアント側) XML アウトプット・アダプタ

アダプタのタイプ： xml_sockout_out。 ソケット (クライアント側) XML アウトプット・アダプタは、送信ネットワーク・アダプタに Event Stream Processor フォーマットでデータを送信します。

アダプタは別のプログラムとの接続を開始し、データを送信します。 接続が切断されると、アダプタは接続をリトライします。

このアダプタを設定して、ストリームのベース状態のみを送信できます。 アダプタはデータを一度送信し、終了しますが、後で再起動できます。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは xml_sockout_out です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Server	**host**	`string`	(必須) サーバ・ホスト名。 デフォルト値は、localhost です。
Port	**port**	`int`	(必須) サーバ・ポート。 **port** が -1 に設定された場合、アダプタはエフェメラルポート・ファイルから読み込みます。デフォルト値は 12345 です。
Include Base Content	**outputBase**	`boolean`	(オプション) ストリームの更新内容だけでなく、初期内容も記録する。デフォルト値は false です。
Ephemeral Port File	**epFile**	`filename`	(詳細) **port** が -1 の場合、サーバのポート番号を格納するファイル。デフォルト値はありません。
Retry period, s	**retryperiod**	`uint`	(詳細) 送信接続を再確立しようとする間隔 (秒単位)。デフォルト値は 1 です。
Only Base Content	**onlyBase**	`boolean`	(詳細) ストリームの初期内容のみを送信する。デフォルト値は false です。
Date Format	**dateFormat**	`string`	(詳細) 日付値を解析するためのフォーマット文字列。 デフォルト値は、%Y-%m-%dT%H:%M:%S です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプ値を解析するためのフォーマット文字列。 デフォルト値は、%Y-%m-%dT%H:%M:%S です。
Field Mapping	**permutation**	`permutation`	(詳細) プラットフォーム固有のフィールドと外部フィールドとの間のマッピング。 デフォルト値は、%Y-%m-%dT%H:%M:%S です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

ソケット (サーバ側) XML インプット・アダプタ

アダプタのタイプ： xml_sockin_in。ソケット (サーバ側) XML インプット・アダプタは、受信ネットワーク・アダプタから Event Stream Processor フォーマットでデータを受信します。

別のプログラムが接続を開始し、データを送信します。

このアダプタを設定して、ストリームのベース状態のみを送信できます。また、繰り返し再接続できます。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは xml_sockin_in です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Port	**port**	`int`	(必須) サーバ・ポート。 **port** が -1 に設定された場合、アダプタはエフェメラルポート・ファイルから読み込みます。デフォルト値は 12345 です。
Match Stream Name	**matchStreamName**	`boolean`	(オプション) true の場合、XML 要素名がストリーム名に対してマッチングされる。 一致しないメッセージは、破棄されます。デフォルト値は false です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Ephemeral Port File	**epFile**	`filename`	(詳細) **port** が -1 の場合、サーバのポート番号を格納するファイル。デフォルト値はありません。
Initial Listen Period (秒)	**retryperiod**	`uint`	(詳細) 継続状態に切り替えるまでに、最初の受信接続を待機する時間の長さを指定する。デフォルト値は 0 です。
Enter Initial State	**initial**	`choice`	(詳細) アダプタがいつ初期ロード状態に入るかを指定する。デフォルト値は never です。
Convert to Safe Opcodes	**safeOps**	`boolean`	(詳細) opcode の INSERT と UPDATE を UPSERT に変換し、DELETE を SAFEDELETE に変換する。デフォルト値は false です。
Skip Deletes	**skipDels**	`boolean`	(詳細) DELETE または SAFEDELETE の opcode を含むローをスキップする。デフォルト値は false です。
Date Format	**dateFormat**	`string`	(詳細) 日付値を解析するためのフォーマット文字列。デフォルト値は、%Y-%m-%dT%H:%M:%S です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプ値を解析するためのフォーマット文字列。デフォルト値は、%Y-%m-%dT%H:%M:%S です。
Block Size	**blockSize**	`int`	(詳細) 擬似トランザクションになる、ブロックあたりのレコード数。デフォルト値は 1 です。
Field Mapping	**permutation**	`permutation`	(詳細) プラットフォーム固有のフィールドと外部フィールドとの間のマッピング。デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

既知の制限事項：

- アダプタはファイル・エントリのストリーム名を無視する。
- すべてのデータは、同じストリームに送信される。
- 同時にサポートされるネットワーク接続は 1 つのみ。

ソケット (サーバ側) XML アウトプット・アダプタ

アダプタのタイプ： xml_sockin_out。 ソケット (サーバ側) XML アウトプット・アダプタは、送信ネットワーク・アダプタから Event Stream Processor フォーマットでデータを受信します。

別のプログラムが接続を開始し、出力アダプタからデータを受信します。

このアダプタを設定して、ストリームのベース状態のみを送信できます。 ストリームのベース状態を送信した後、ソケットは閉じますが、繰り返し再接続できます。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは xml_sockin_out です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Port	**port**	`int`	(必須) サーバ・ポート。 **port** が -1 に設定された場合、アダプタはエフェメラルポート・ファイルから読み込みます。デフォルト値は 12345 です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Include Base Content	**outputBase**	`boolean`	(オプション) ストリームの更新内容だけではなく、初期内容も記録して開始する。デフォルト値は false です。
Ephemeral Port File	**epFile**	`filename`	(詳細) **port** が -1 の場合、サーバのポート番号を格納するファイル。デフォルト値はありません。
Only Base Content	**onlyBase**	`boolean`	(詳細) アダプタは 1 回だけストリームの初期内容を送信する。デフォルト値は false です。
Date Format	**dateFormat**	`string`	(詳細) 日付値を解析するためのフォーマット文字列。 デフォルト値は、%Y-%m-%dT%H:%M:%S です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプ値を解析するためのフォーマット文字列。 デフォルト値は、%Y-%m-%dT%H:%M:%S です。
Field Mapping	**permutation**	`permutation`	(詳細) プラットフォーム固有のフィールドと外部フィールドとの間のマッピング。 デフォルト値はありません。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

既知の制限事項：

- 一度に 1 つだけネットワーク接続をサポートする。

ソケット (サーバ側) CSV インプット・アダプタ

アダプタのタイプ： dsv_sockin_in。ソケット (サーバ側) CSV インプット・アダプタは、受信ネットワーク・アダプタから Event Stream Processor の区切られたフォーマットでデータを受信します。

別のプログラムが接続を開始し、アダプタにデータを送信します。

データにヘッダを設定しないこともできますし、ヘッダにフィールド名を指定しないこともできます。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは dsv_sockin_in です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Port	**port**	`int`	(必須) サーバ・ポート。 **port** が -1 に設定された場合、アダプタはエフェメラルポート・ファイルから読み込みます。デフォルト値は 12345 です。
Stream name、opcode expected	**expectStreamNameOpcode**	`boolean`	(オプション) true の場合、最初の 2 つのフィールドはそれぞれ、ストリーム名と opcode と解釈される。 ストリーム名に一致しないメッセージは、破棄されます。デフォルト値は false です。
Delimiter	**delimiter**	`string`	(詳細) カラムを区切るために使用する記号。 デフォルト値は、カンマ (,) です。
Has Header	**hasHeader**	`boolean`	(詳細) 最初の行がフィールドの説明を含むかどうかを決定する。デフォルト値は false です。
Ephemeral Port File	**epFile**	`filename`	(詳細) **port** が -1 の場合、サーバのポート番号を格納するファイル。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Initial Listen Period (秒)	**retryperiod**	`uint`	(詳細) 継続状態に切り替えるまでに、最初の受信接続を待機する時間の長さ。デフォルト値は 0 です。
Enter Initial State	**initial**	`choice`	(詳細) アダプタがいつ初期ロード状態に入るかを指定する。 デフォルト値は never です。
Convert to Safe Opcodes	**safeOps**	`boolean`	(詳細) opcode の INSERT と UPDATE を UPSERT に変換し、DELETE を SAFEDELETE に変換する。デフォルト値は false です。
Skip Deletes	**skipDels**	`boolean`	(詳細) DELETE または SAFEDELETE の opcode を含むローをスキップする。デフォルト値は false です。
Date Format	**dateFormat**	`string`	(詳細) 日付値を解析するためのフォーマット文字列。 デフォルト値は、%Y-%m-%dT%H:%M:%S です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプ値を解析するためのフォーマット文字列。 デフォルト値は、%Y-%m-%dT%H:%M:%S です。
Block Size	**blockSize**	`int`	(詳細) 擬似トランザクションになる、ブロックあたりのレコード数。デフォルト値は 1 です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Field Mapping	**permutation**	`permutation`	(詳細) プラットフォーム固有のフィールドと外部フィールドとの間のマッピング。 デフォルト値はありません。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

既知の制限事項：

- ファイル・ローのストリーム名は無視される。
- すべてのデータは、同じストリームに送信される。
- 1 つだけネットワーク接続をサポートする。

ソケット (サーバ側) CSV アウトプット・アダプタ

アダプタのタイプ： dsv_sockin_out。 ソケット (サーバ側) CSV アウトプット・アダプタは、受信ネットワーク・アダプタに Event Stream Processor の区切られたフォーマットでデータを送信します。

アダプタを設定して、ストリームのベース状態のみを送信できます。 ストリームのベース状態を送信した後、ソケットは閉じますが、繰り返し再接続できます。

データにヘッダを設定しないこともできますし、ヘッダにフィールド名を指定しないこともできます。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは dsv_sockin_out です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Port	**port**	`int`	(必須) サーバ・ポート。**port** が -1 に設定された場合、アダプタはエフェメラルポート・ファイルから読み込みます。デフォルト値は 12345 です。
Include Base Content	**outputBase**	`boolean`	(オプション) ストリームの更新内容だけでなく、初期内容も記録する。デフォルト値は false です。
Prepend Stream Name, Opcode	**prependStreamNameOpcode**	`boolean`	(オプション) true の場合、各メッセージはストリーム名と opcode で開始する。デフォルト値は false です。
Ephemeral Port File	**epFile**	`filename`	(詳細) **port** が -1 の場合、サーバのポート番号を格納するファイル。デフォルト値はありません。
Only Base Content	**onlyBase**	`boolean`	(詳細) アダプタは 1 回だけストリームの初期内容のみを送信する。デフォルト値は false です。
Delimiter	**delimiter**	`string`	(詳細) カラムを区切るために使用する記号。 デフォルト値は、カンマ (,) です。
Has Header	**hasHeader**	`boolean`	(詳細) 最初の行がフィールドの説明を含むかどうか。デフォルト値は false です。
Date Format	**dateFormat**	`string`	(詳細) 日付値を解析するためのフォーマット文字列。デフォルト値は、%Y-%m-%dT%H:%M:%S です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプ値を解析するためのフォーマット文字列。 デフォルト値は、%Y-%m-%dT%H:%M:%S です。
Field Mapping	**permutation**	`permutation`	(詳細) プラットフォーム固有のフィールドと外部フィールドとの間のマッピング。 デフォルト値はありません。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

既知の制限事項：

- 1 つだけネットワーク接続をサポートする。

SMTP アウトプット・アダプタ

アダプタのタイプ： smtp_out。 SMTP アウトプット・アダプタは、ストリーム・レコードで構成される電子メールを送信します。

レコードごとに、電子メール本体は以下で構成されます。

- ストリーム名
- カラム名と値

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは smtp_out です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
SMTP Host	**smtpHost**	`string`	(必須) 電子メール・サーバの名前または IP アドレス。 デフォルト値はありません。
fromAddress	**from**	`string`	(必須) 送信者の電子メール・アドレス。 デフォルト値はありません。
Importance Column	**importanceColumn**	`string`	(必須) 電子メールの重要度が格納されているストリーム・カラムの名前。重要度の有効な値は次のとおりです。 high、normal、low。 デフォルト値は importance です。 値は、大文字、小文字が区別されます。
Address Column	**addressColumn**	`string`	(必須) セミコロンで区切られた受信者の電子メール・アドレスのリストが格納されているカラムの名前。 デフォルト値はありません。
Subject Column	**subjectColumn**	`string`	(必須) 電子メールのサブジェクトが格納されているストリーム・カラムの名前。 デフォルト値はありません。
SMTP Port	**smtpPort**	`uint`	(オプション) SMTP サーバによって使用されるポート。デフォルト値は 25 です。
SMTP Username	**smtpUsername**	`string`	(オプション) このプロパティを設定すると、SMTP 認証によって SMTP パスワードの設定も要求される。
SMTP Password	**smtpPassword**	`string`	(オプション) SMTP Username を設定した場合はこのプロパティも設定。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Use SSL	**useSSL**	`boolean`	(オプション) このオプションを有効にするには、セキュリティ証明書を Java キーストア・ファイル (`install/lib/jre/lib/security/cacerts` の下に存在) にインポートすることが必要。 セキュリティ証明書を設定した後に、このオプションを有効にできます。 デフォルト値は false です。 **重要：** SSL 用の SMTP サーバを有効にしてから、このオプションを設定してください。
Use TLS	**useTLS**	`boolean`	(オプション) このオプションを有効にするには、セキュリティ証明書を Java キーストア・ファイル (`install/lib/jre/lib/security/cacerts` の下に存在) にインポートすることが必要。 セキュリティ証明書を設定した後に、このオプションを有効にできます。 デフォルト値は false です。 **重要：** TLS 用の SMTP サーバを有効にしてから、このオプションを設定してください。
cc Column	**ccColumn**	`string`	(詳細) セミコロンで区切られた受信者の cc 電子メール・アドレスのリストが格納されているカラムの名前。 デフォルトでは、cc 電子メールは送信されません。
bcc Column	**bccColumn**	`string`	(詳細) セミコロンで区切られた受信者の bcc 電子メール・アドレスのリストが格納されているカラムの名前。 デフォルトでは、bcc 電子メールは送信されません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Column Names	**columnNames**	`string`	(詳細) 電子メールに値があるストリーム・カラムの名前のコロン区切りのリスト。 デフォルトでは、電子メールには、ストリームのすべてのカラムの値があります。
Show Column Names	**showColumnNames**	`boolean`	(詳細) true の場合、アダプタには、電子メールのカラム名が値と共にあります。 false の場合、値のみがあります。デフォルト値は true です。
Number of Resend Attempts	**resendAttempts**	`integer`	(詳細) 電子メールの最初の送信が失敗した場合に、再送信を試行する回数。 デフォルトは 0 で、電子メールの再送信試行は行われません。 このプロパティには、0 ～ 10 の範囲で適度な値を選択します。 電子メールの再送信試行回数を大きい値に設定すると、特にネットワーク問題によって送信の失敗が多発している場合や、大量のレコードが電子メールによる送信待ちになっている場合に、大量のメモリが消費されることがあります。
Log Alert	**logAlert**	`boolean`	(詳細) true の場合、電子メールの送信が成功するか、失敗するごとに、デバッグ・レベル 1 で警告がログ記録される。 デフォルト値は true です。
Date Format	**dateFormat**	`string`	(詳細) 日付のフォーマット。 デフォルト値は、`%Y-%m-%dT%H:%M:%S` です。
Timestamp Format	**timestampFormat**	`string`	(詳細) タイムスタンプのフォーマット。 デフォルト値は、`%Y-%m-%dT%H:%M:%S` です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

既知の制限事項：

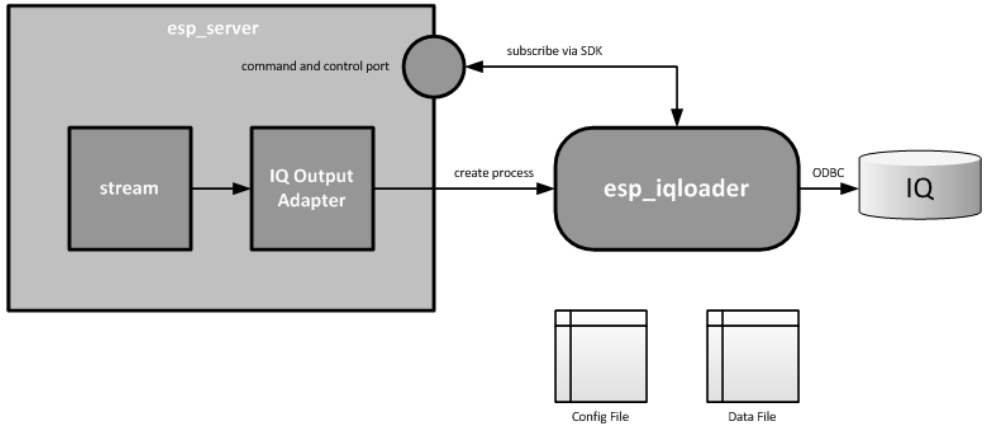
- Microsoft Outlook® を使用している場合、以下の操作を行って、余分な改行を削除する機能を無効にする。
  1. Outlook を開き、[ツール] > [オプション] を選択します。
  2. [初期設定] タブで [メール オプション] を選択します。
  3. [テキスト形式メッセージ内の余分な改行を削除する] チェック・ボックスをオフにします。 [OK] を 2 回クリックします。

Sybase IQ アウトプット・アダプタ

アダプタのタイプ： sybase_iq_out。 Sybase IQ アウトプット・アダプタは、Event Stream Processor から Sybase IQ にデータをロードします。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは sybase_iq_out です。

Sybase IQ アウトプット・アダプタは、**esp_iqloader** ユーティリティを使用して Sybase IQ にデータをロードします。 アダプタが起動するときに、アダプタは、設定ファイルで指定したストリームにサブスクライブする **esp_iqloader** ユーティリティを起動し、ODBC を使用してそれらのストリームから IQ にデータをロードします。



プロパティ・ラベル	プロパティ ID	タイプ	説明
Configuration File	**configFile**	`filename`	(必須) Sybase IQ 接続の詳細、ストリーム情報、ロード・オプションを提供する **esp_iqloader** 設定ファイルへのフル・パス。 このファイルのフォーマットの詳細については、『ユーティリティ・ガイド』を参照してください。
User Name	**user**	`string`	(オプション) **esp_iqloader** ユーティリティが Event Stream Processor に接続するときに使用するユーザ名。 **注意：** このプロパティは、サーバ RSA、ユーザ名またはパスワード、Kerberos、LDAP の認証を使用するときに必要です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Password	**password**	`password`	(オプション) Event Stream Processor に接続するために使用するパスワード。 **注意：** RSA キー・ファイルを指定する場合、必要ありません。
RSA Key File	**rsaKeyFile**	`RuntimeFilename`	(オプション) RSA キーのプライベート・キー・ファイルへのフル・パス。 Event Stream Processor へのサーバ RSA 認証を使用するときに、このパスを指定します。 **注意：** ユーザ名設定してから、このオプションを設定します。
Use Kerberos	**useKerberos**	`boolean`	(オプション) true の場合、RSA キー・ファイル・パラメータが指定されていないときに、Event Stream Processor への Kerberos 認証を使用する。 RSA キー・ファイル・パラメータが指定されていて、**UseKerberos** が true に設定されている場合、RSA 設定は Kerberos 設定に優先し、RSA 認証が使用されます。 デフォルト値は false です。 **注意：** ユーザ名設定してから、このオプションを設定します。

プロパティ・ラベル	プロパティ ID	タイプ	説明
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。デフォルト値はありません。
Archive Deltas	**archiveDeltas**	`boolean`	(詳細) true の場合、デルタをアーカイブする。それ以外の場合、データのスナップショットのみがアーカイブされます。 デルタ・モードでは、指定したストリームまたはウィンドウから Sybase IQ に、すべてのデータ (既存と更新) が継続的にアップロードされます。 スナップショット・モードでは、アダプタの起動時と終了時に、すべての既存のデータが指定したストリームまたはウィンドウからアップロードされます。 既存のデータとは、アダプタが起動、接続したときに、ウィンドウに既に存在するデータを指します。デフォルト値は true です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Project URI	**projectUri**	`string`	(必須) Event Stream Processor クラスタのプロジェクトに接続するために必要な URI。たとえば、`<hostname>:<port>/<workspace-name>/<project-name>` のように記述します。
Swap Bytes	**byteSwap**	`boolean`	(詳細) Event Stream Processor と **esp_iqloader** ユーティリティを異なるアーキテクチャ (リトル・エディアン／ビッグ・エディアン) で実行する場合、true に設定する。デフォルト値は false です。
Recover Only	**recoverOnly**	`boolean`	(詳細) true の場合、Event Stream Processor から読み込まれたが、アーカイブされていないデータをリカバリし、終了する。デフォルト値は false です。
Datawarehousing Mode	**dataWarehouse**	`boolean`	(詳細) true の場合、更新を挿入として扱い、削除を無視する。デフォルト値は false です。
Archive Interval	**archiveInterval**	`uint`	(詳細) 1 つのセットのデータをアーカイブし、次のセットのデータをアーカイブするまで、待機する時間の長さを秒単位で指定する。デフォルト値は 1 です。
Precision	**precision**	`uint`	(詳細) money と float のデータ型を使用するための精度を指定する 0 ～ 6 の値。デフォルト値は 6 です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Subscription Buffer Size	**queueSize**	`uint`	(詳細) 使用するサブスクリプション・バッファ・サイズを指定する 1000 より大きい正の値。 **注意：** データが集中し、サブスクリプション・バッファが満杯であると Event Stream Processor がレポートする場合、この値を変更してください。
Commit Batch Size	**batchSize**	`uint`	(詳細) データを Sybase IQ にアーカイブするために使用するバッチ・サイズ。 デフォルト値は 1000 です。 **重要：** バッチ・サイズ設定ファイルにバルク・ロードを指定します。
ODBC Retry Attempts	**odbcRetryTimes**	`uint`	(詳細) ODBC 接続ができないときに、接続をリトライする回数。デフォルト値は 5 です。
ODBC Retry Interval	**odbcRetryInterval**	`uint`	(詳細) ODBC 接続をリトライするまでに待機する秒数。デフォルト値は 60 です。

既知の制限事項：

- Sybase IQ アウトプット・アダプタをストリームにアタッチしても、アダプタがデータをこのストリームからアーカイブする保証はない。 アダプタは指定した設定ファイルを使用して、この情報を取得します。 設定ファイルによって、異なるストリームが指定されている場合、そのストリームがアーカイブされます。

Sybase IQ アダプタのデータ型マッピング

Event Stream Processor のデータ型は、Sybase IQ のデータ型にマップされます。

必要に応じて、Sybase IQ にテーブルを作成し、Event Stream Processor のデータ型を他の互換性のある Sybase IQ のデータ型にマップできます。 以下のデータ型マッピングは、デフォルトのマッピングです。

注意： 日付とタイムスタンプの Event Stream Processor データ型は、`datetime` Sybase IQ データ型のみと互換性があります。

Event Stream Processor のデータ型	Sybase IQ データ型
`bigdatetime`	`timestamp`
`binary`	`binary`
`boolean`	`varchar(5)`
`date`	`datetime`
`float`	`float`
`integer`	`integer`
`interval`	`bigint`
`long`	`bigint`
`money`	`decimal(38,4)`
`money(n)`	`decimal(38,n)`
`string`	`varchar(n)`
`timestamp`	`datetime`

WebSphere MQ アダプタ

Event Stream Processor では、WebSphere MQ キューと Event Stream Processor ストリームの間での読み込みと書き込みを可能にする WebSphere MQ アダプタを使用できます。

Event Stream Processor では、WebSphere MQ サーバによる Event Stream Processor エンジンに対する読み込みと書き込みができます。 必要に応じて、この内部アダプタをカスタマイズできます。 WebSphere MQ メッセージは構造化されないため、適宜スキーマを定義して、MQ メッセージを準備します。 スキーマ定義では、す

べての Event Stream Processor データ型を使用できます。 バイナリ・データ、文字列などを Event Stream Processor エンジンとの間で送受信できます。

MQ クライアント・アダプタに MQ 7.0.1 クライアント・ソフトウェアがインストールされていることを確認します。 アダプタがインストールされているソフトウェアと一致しない場合、エラーが発生します。Sybase WebSphere MQ アダプタは、サーバと同じホスト・コンピュータで WebSphere MQ クライアント・ソフトウェアを使用するように設計されています。 ただし、WebSphere MQ サーバは同じコンピュータにも別のコンピュータにも配置できます。

WebSphere MQ インプット・アダプタと WebSphere MQ アウトプット・アダプタは、Event Stream Processor と WebSphere クエリとの間でのデータの挿入、削除、更新、アップサートの opcode をサポートします。

WebSphere MQ インプット・アダプタ

アダプタのタイプ： wsmq_in。デフォルトの WebSphere MQ インプット・アダプタは CSV フォーマットの文字列を読み込みます。

メッセージ内のデータの順序が、アダプタをアタッチする入力ストリームのスキーマと一致するようにします。 WebSphere MQ インプット・アダプタは、ストリーム名と opcode 命令 (INSERT、DELETE、UPDATE、UPSERT) をキューから引き出す CSV データに適用します。 **expectStreamNameOpcode** プロパティを参照してください。

注意： WebSphere MQ インプット・アダプタを使用して WSMQ からデータを読み込むときに、WebSphere MQ アウトプット・アダプタではなく JMS プログラムを使用してデータをパブリッシュする場合、次の行を JMS プログラムに設定します。

```
queue.setTargetClient(com.ibm.mq.jms.JMSC.MQJMS_CLIENT_NONJMS_MQ);.
```

Linux インストール環境や UNIX インストール環境で正常にアダプタを実行するには、LD_LIBRARY_PATH を設定して MQ クライアント・ライブラリを参照します。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは wsmq_in です。

注意： サーバに障害が発生した場合にデータを消失しないように、Sybase ではログ・ストアを入力ウィンドウにアタッチすることをおすすめします。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Queue Name	**QueueName**	`string`	(必須) メッセージの送信対象となる、サーバ上のキューの名前。 このキューは、Queue Manager Name で指定されたキュー・マネージャによって管理される必要があります。 デフォルト値はありません。
Queue Manager Name	**QueueManagerName**	`string`	(必須) メッセージの送信対象となる、サーバ上のキュー・マネージャの名前。 デフォルト値はありません。
MQ System Name	**SystemName**	`string`	(必須) MQ サーバ・システムの名前。 この名前には記号名または IP アドレスを指定できます。 デフォルト値はありません。
Port	**Port**	`string`	(必須) MQ サーバ・キュー・リスナがアタッチされる MQ サーバ・システムのポート番号。 デフォルト値はありません。
MQ Channel	**Channel**	`string`	(必須) キューに関連付けられた MQ サーバ・チャネルの名前。 デフォルト値はありません。
Maximum Inpu Buffer Sizet	**MaxBufferSize**	`uint`	(必須) バッファの最大サイズ (バイト単位)。 デフォルト値はありません。
Sync Point Commit Mode	**syncpointmode**	`boolean`	(オプション) 同期ポイント・コミット・モードを有効にする。 このプロパティ値を設定し、アダプタからサーバへのメッセージの配信を保証します。デフォルト値は false です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Batch Size	**batchsize**	`uint`	(オプション) 同期点コミット・モードでコミットするバッチのレコード数を指定する。デフォルト値は 1 です。
CSV Delimiters	**CsvSeparators**	`string`	(オプション) CSV フィールド・セパレータ。複数の文字を指定できます。デフォルト値は、カンマ (,) です。
CSV Escape Characters	**CsvEscapeChars**	`string`	(オプション) デリミタ、エスケープ文字、引用符文字などの特殊文字の意味をエスケープする文字。複数の文字を指定できます。 デフォルト値は、円記号 (¥) です。
CSV Quote Characters	**CsvQuoteChars**	`string`	(オプション) フィールドの開始と終了の境界を示す文字。 デフォルト値は、二重引用符 (") です。
Perform CSV Trimming	**CsvTrimming**	`boolean`	(オプション) true に設定すると、各フィールドの先頭と末尾からホワイトスペースをトリムする。 true の場合、スペースのみが存在する引用符で囲まれたフィールドは、NULL として解釈されます。 デフォルト値は true です。
Stream Name, Opcode Expected	**TimestampColumnFormat**	`string`	(オプション) タイムスタンプ値のフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Timestamp Column Format	**expectStreamNameOpcode**	`boolean`	(詳細) true の場合、CSV レコードの先頭の 2 つのフィールドはストリーム名と opcode として解釈される。デフォルト値は false です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Date Column Name	**DateColumnName**	`string`	(詳細) 日付値がファイルに保存されるフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

参照：
- *第 5 章、「保証された配信」* (591 ページ)

WebSphere MQ アウトプット・アダプタ

アダプタのタイプ： wsmq_out。 デフォルトの WebSphere MQ アウトプット・アダプタは CSV フォーマットの文字列をパブリッシュします。

WebSphere MQ アダプタでは、アダプタにパブリッシュするストリームのスキーマがフィールドの順序とデータ型を決定するため、ヘッダ行が生成されません。 カラムは、適切なデータ型のデフォルトの表示フォーマットでパブリッシュされます。 アダプタは、キューに追加される CSV データの先頭にストリーム名と opcode 命令 (INSERT、DELETE、UPDATE、UPSERT) を付加します。 **prependStreamNameOpcode** プロパティを参照してください。

Linux インストール環境や UNIX インストール環境で正常にアダプタを実行するには、LD_LIBRARY_PATH を設定して MQ クライアント・ライブラリを参照します。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは wsmq_out です。

注意： このアダプタは、TCP/IP を使用して転送します。 他のプロトコルを使用するには、そのプロトコルの適切な設定とインタフェース・プロパティを指定してください。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Queue Name	**QueueName**	`string`	(必須) メッセージの送信対象となる、サーバ上のキューの名前。 このキューは、Queue Manager Name で指定されたキュー・マネージャによって管理される必要があります。 デフォルト値はありません。
Queue Manager Name	**QueueManagerName**	`string`	(必須) メッセージの送信対象となる、サーバ上のキュー・マネージャの名前。 デフォルト値はありません。
MQ System Name	**SystemName**	`string`	(必須) MQ サーバ・システムの名前。 この名前には記号名または IP アドレスを指定できます。 デフォルト値はありません。
Port	**Port**	`string`	(必須) MQ サーバ・キュー・リスナがアタッチされる MQ サーバ・システムのポート番号。 デフォルト値はありません。
MQ Channel	**Channel**	`string`	(必須) キューに関連付けられた MQ サーバ・チャネルの名前。 デフォルト値はありません。
CSV Field Separator	**CsvSeparatorChar**	`string`	(オプション) CSV フィールド・セパレータ。1 文字を指定します。 デフォルト値は、カンマ (,) です。
CSV Escape Character	**CsvEscapeChar**	`string`	(オプション) フィールド・セパレータ、エスケープ文字、引用符文字などの特殊文字の意味をエスケープする文字。 デフォルト値は、円記号 (¥) です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
CSV Quote Character	**CsvQuoteChar**	`string`	(オプション) 任意の文字で構成されるフィールドの開始と終了の境界を示す文字。埋め込み引用符文字はすべてエスケープされます。 デフォルト値は、二重引用符 (") です。
Prepend Stream Name, Opcode	**prependStreamNameOpcode**	`boolean`	(詳細) true の場合、すべての CSV レコードにはストリーム名と opcode が先頭に付加される。デフォルト値は false です。
Timestamp Column Format	**TimestampColumnFormat**	`string`	(詳細) タイムスタンプ値のフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Date Column Format	**DateColumnFormat**	`string`	(詳細) 日付値がファイルに保存されるフォーマット。 デフォルト値は、YYYY-MM-DDTHH:MM:SS.SSS です。
Runs Adapter in GD Mode	**enableGDMode**	`boolean`	(詳細) true に設定すると、アダプタが、保証された配信 (GD) モードで動作し、GD 関連のすべてのパラメータが必須になる。デフォルト値は false です。
Name of Column Holding GD Key	**gdKeyColumn**	`string`	(詳細) GD キーを保持するフレックス演算子のカラム名を指定する。 GD キーは、対象となるストリームに格納される opcode に関係なく、各イベントを一意に識別する、常に増加する値です。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Name of Column Holding opcode	**gdOpcodeColumn**	`string`	(詳細) opcode を保持するフレックス演算子のカラム名を指定する。 opcode は、対象のストリームで発生するイベントのオペレーション・コード (たとえば、挿入、更新、または削除) です。 デフォルト値はありません。
Name of Truncate Stream	**gdControlStream**	`string`	(詳細) GD セットアップの制御ウィンドウの名前を指定する。 制御ウィンドウはソース・ストリームで、そのデータがアダプタによって処理済みで、安全に削除できることをフレックス演算子に通知します。 デフォルト値はありません。
Purge After Number of Records	**gdPurgeInternal**	`int`	(詳細) レコード数を指定し、この数に到達すると、フレックス演算子はパージを実行する。 デフォルト値は 1000 です。
Batch Size to Update Truncate Stream	**gdBatchSize**	`int`	(詳細) レコード数を指定し、この数に到達すると、制御ウィンドウは最新の GD キーで更新される。 デフォルト値は 1000 です。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

参照：
- *第 5 章、「保証された配信」* (591 ページ)

キュー設定
このサンプル・コードを使用して、キュー・マネージャ、サーバ・キュー、チャネル、リスナを呼び出します。

標準の MQ サーバ設定では、以下が提供されます。

- `queue.manager.1` と呼ばれるデフォルト・キュー・マネージャ
- `QUEUE1` と呼ばれるローカル・サーバ・キュー
- `channel1` と呼ばれるサーバ接続チャネル
- TCP/IP ポート 2001 上の `listener1` と呼ばれるリスナ

```
Create a default queue manager called queue.manager.1 and
start it:
crtmqm -q queue.manager.1
strmqm
dspmq # display list of active queues
Now create a local queue, a channel and a listener:
runmqsc
define qlocal(QUEUE1)
5
define channel (channel1) chltype (svrconn) trptype (tcp)
¥
mcauser ('mqm')
define listener (listener1) trptype (tcp) control (qmgr) ¥
port (2001)
start listener (listener1)
end
```

注意： この設定例では、スペースの制約があり、読みやすさを考慮してバックスラッシュを使用しています。 システムでキューを設定するときは、この情報を 1 行で記述してください。

外部アダプタ

Event Stream Processor には、非標準のインタフェースからの非標準データ・フォーマットを処理するために、特別な外部アダプタが用意されています。

外部アダプタは、サーバと同じマシンまたは異なるマシン上で個別のプロセスとして実行します。 これらのアダプタは、サーバと関連するクエリ・モジュールから独立して起動、停止します。

参照：
- *アダプタ・パラメータのデータ型* (9 ページ)

- *スキーマ検出をサポートするアダプタ* (585 ページ)
- *カスタム外部アダプタ* (562 ページ)

ESP Add-In for Microsoft Excel

Sybase ESP Add-in for Microsoft Excel® は、Microsoft Excel 用のリアルタイム・データ・アドインで、実行中の 1 つ以上の Event Stream Processor プロジェクトからのレコードの表示と取得、レコードのそれらへのパブリッシュを可能にします。

ESP Add-in for Microsoft Excel は、Linux または Solaris のプラットフォームをサポートしません。 Windows の 32 ビットと 64 ビットのエディションで実行でき、MS Excel 2007 と 2010 (32 ビット・エディションのみ) と互換性があります。

表示側では、SybaseRT を使用することによって、ストリームを選択し、それらのストリーム内のカラムを表示できます。 データ値に基づいてレコードをフィルタし、直近の "N" レコードを表示できます。または、指定したフィルタに一致する直近の "N" レコードを表示できます ("N" は指定可能なレコード数です)。

パブリッシュ側では、SybaseRT を使用することによって、選択したセルの範囲内でデータが変更されたときには常に、データが自動的にパブリッシュされるようにできます。 また、セルの範囲を選択し、パブリケーション作成ウィザードを使用することによって、手動でデータを Event Stream Processor にパブリッシュできます。

接続ウィザード

接続ウィザードを使用すると、1 つ以上の Event Stream Processor インスタンスに同時に接続できます。

Excel ワークブックの接続情報は、ワークブットと共に保存されます。

コンポーネント	説明
Connections	新しい接続の名前を入力するか、以前に定義された接続のリストから選択。 接続を選択すると、その接続に関連付けられているすべての情報が表示されます。
Host	(必須) 接続先の Event Stream Processor クラスタ・マネージャのホスト名を入力。
Port	(必須) Event Stream Processor クラスタ・マネージャのポートを入力。
Workspace	そのプロジェクトが一部分を構成するクラスタを入力。
Project	接続先のプロジェクトを入力。

コンポーネント	説明
User	クラスタ・マネージャに接続するためのユーザ名を入力。 このプロパティは、認証タイプとして none を選択していないかぎり、必須です。
Password	(オプション) ユーザ名に関連付けられているパスワードを入力。
SSL	SSL モードを有効にして起動された Event Stream Processor クラスタ・マネージャに接続するには、これを使用。
Authentication Type	リストから認証タイプを選択。 選択する認証タイプは、Event Stream Processor を起動するときに使用したモードと一致する必要があります。
RSA Key File	RSA 認証が選択されている場合、RSA キー・ファイルを入力。 キー・ファイルのロケーションと名前を入力するか、フィールドの隣のボタンをクリックして参照し、ファイルを選択するかのいずれかを実行できます。
Save	アクティブなワークブックに関連付けられている非表示のワークシートの接続情報を保存。 これを使用すると、アクティブなワークブックを再びオープンしたときに、接続情報が自動的に取得され、表示されます。
Connect	すべての情報の入力後に、このボタンをクリックしてサーバに接続。 接続が正常に確立されると、この接続では、[Disconnect] ボタンのみが利用できるようになります。 また、接続情報が今後の使用のために保存されます。
Disconnect	Event Stream Processor への接続を切断するには、このボタンをクリック。 切断が正常に行われると、このボタンは無効になり、[Connect] と [Delete] のボタンが有効になります。 この接続を使用しているクエリがアクティブな場合、ユーザに切断の確認が求められ、ユーザが確認すると、クエリは停止されます。
Delete	接続を削除するには、このボタンをクリック。
Hide	すべての情報を維持しながらウィンドウを非表示にするには、このボタンをクリック。画面を再び表示するには、ツールバーの [SybaseRT] ボタンをクリックします。

サブスクリプション・ウィザード

[Subscription Wizard] ペインを使用すると、1 つ以上のサブスクリプションの定義と制御を行い、結果を Excel ワークシートに表示できます。

SybaseRT は、クエリのロケーションを追跡します。 これは、定義されているセルが水平方向または垂直方向に移動しても、クエリが動作することを意味します。

コンポーネント	説明
Subscription Queries	新しいクエリの名前を入力するか、ドロップ・ダウン・リストからクエリ名を選択して、以前に定義したクエリを選択します。 以前に保存したクエリを選択すると、選択したクエリに関連付けられているすべての情報が自動的に表示されます。
Connection Name	現在アクティブな接続のドロップ・ダウン・リストからクエリに関連付けられている接続を選択。 クエリは、関連付けられている接続がアクティブな場合のみ、実行できます。
Start Cell	リアルタイム・データ式の挿入を開始するための、Excel ワークシートのロケーションを入力。 ロケーションを "A1" 表記で指定します。 たとえば、B5 の値を入力すると、SybaseRT は式をカラム B、ロー 5 で始まるグリッドに挿入します。
Max Rows	Excel ワークシートで表示するレコード数 (最大許容値は 65536) を入力。 指定した数よりも多くのローが存在する場合、新しい方から順に表示されます。
Get Base Transactions	ストリーム内の基準レコードを取得してから新しいトランザクションを取得するには、このチェック・ボックスをオンにする。 新しいトランザクションだけを取得する場合は、このチェック・ボックスをオフのままにします。 トランザクション数が相対的に少ない小さなテーブルでは、このチェック・ボックスをオンにすることをおすすめします。 そうしないと、クエリのデータが表示されません。 トランザクションのアクティビティが高い動的テーブルでは、このチェック・ボックスをオフのままにすることをおすすめします。 オンにすると、Excel が、起動するたびに数百万件のレコードをロードしようとする可能性があります。
Streams	選択した接続に関連付けられているサーバで利用可能なすべてのストリームを表示。 接続を選択すると、このボックスに値が自動的に設定されます。

コンポーネント	説明
Columns	ストリームの 1 つを選択すると、この領域に各カラムが、そのデータ型とキー・カラムであるかどうかを示すチェック・ボックスと共に表示される。 指定されているキー・カラムとは異なるキー・カラムをストリーム用に選択できます。
SQL Statement	Event Stream Processor から取得されるものをカスタマイズする必要がある場合、このテキスト・ボックスに SQL 文を指定。 文には、ジョイン、Group By 句、Order By 句を指定できません。これは、SQL はストリームの個々のトランザクション・ログに適用され、ストリーム内のデータには適用されないためです。 **esp_query** に関する説明と、サポートされる SQL 構文の情報については、「ユーティリティ・ガイド」を参照してください。 [SQL Statement] テキスト・ボックスは、<<SQL Statement>> エントリが [Streams] リスト・ボックスで選択された場合のみ有効になります。
Parse SQL	[SQL Statement] テキスト・ボックスで入力された SQL 文を解析するには、このボタンをクリック。 SQL が正常に解析されると、カラム名と対応するデータ型が [Columns] リスト・ボックスに表示されます。 デフォルトでは、いずれのカラムもキー・フィールドとしてマーク付けされませんが、リアルタイム・データ・クエリを適用できるようにするには、事前に適切なキー・カラムを選択する必要があります。
Apply	新しいサブスクリプションを設定するか、既存のサブスクリプションを変更した後に、リアルタイム・データ式を Excel ワークシートに適用するには、このボタンをクリック。 式が適用されると [Start] ボタンが表示され、クエリを開始できるようになります。
Reset	サブスクリプション・クエリのプロパティの最後に保存された値を表示するには、このボタンをクリック。 クエリに対して変更を行った後に、完全に元に戻すことが必要になった場合にこのボタンを使用します。
Delete	以前に保存したクエリを削除するには、このボタンをクリック。
Start	クエリを起動するには、このボタンをクリック。 データが Excel ワークシートに表示されます。
Stop	実行中のクエリを停止するには、このボタンをクリック。 Excel ワークシートに以降のデータは表示されません。 ただし、既に表示されているデータは、ワークシートを閉じて再び開くまで、またはクエリを再び起動するまで、表示されたままになります。

パブリケーション作成ウィザード

パブリケーション作成ウィザードを使用すると、データをストリームに手動でパブリッシュしたり、自動パブリッシュのためにパブリッシュ式をグラフィカルに構築したりできます。

コンポーネント	説明
Connection Name	パブリッシュに使用する接続の名前。 アクティブな接続のみが、このボックスに表示されます。 接続をクリックすると、接続オブジェクトの接続先のストリームが [Streams] リスト・ボックスに表示され、ストリームのカラムとデータ型が Columns という名前のテーブルに表示されます。
Operation Code	このリスト・ボックスから、INSERT、UPDATE、DELETE、または UPSERT のいずれかを選択します。 何も選択しないと、デフォルト opcode の UPSERT が適用されます。 **注意：** レコードが存在する場合、UPDATE、DELETE、または UPSERT を選択します。 レコードが存在しない場合は、INSERT を選択します。
Data Range	パブリッシュするデータの存在する、Excel ワークシートでのセルの範囲を指定。 このフィールドは、直接編集できません。パブリッシュするセルをワークシートで選択し、このフィールドの隣の青色のボタンをクリックすることによって設定します。 連続していない複数の領域を同時にパブリッシュすることはサポートされていません。 Excel ワークシートで連続していない複数の領域を選択し、これらの選択した領域のアドレスをこのフィールドで表示すると、データをパブリッシュするときにエラーが発生します。
WorkBook Name	選択した範囲が存在するワークブックを表示する読み込み専用のフィールド。
WorkSheet Name	選択した範囲が存在するワークシートの名前を表示する読み込み専用のフィールド。

コンポーネント	説明
First Row Has Columns	選択した範囲の最初のローはカラム名であることを示すには、このチェック・ボックスをオンにする。 そうでない場合は、オフのままにします。 カラム名を指定する場合、データ・カラムは任意の順序で指定できます。また、該当するフィールドの値のみを指定する必要があります。残りのフィールドには、自動的に null が設定されます。 ただし、カラム名を指定しないと、ストリーム内のすべてのカラムが Event Stream Processor で定義されているのと同じ順序で提供されると SybaseRT によって想定されます。
Transpose Rows To Columns	レコードのデータ・カラムが、通常のレコード表示である複数カラムで水平に配置されるのでなく、垂直に配置されている場合は、このチェック・ボックスをオンにする。 そうでない場合は、オフのままにします。
Streams	選択した接続が接続されている Event Stream Processor のストリームのリストが自動的に維持されるリスト・ボックス。 パブリッシュ先のストリームを選択します。
Columms	自動的に維持されるテーブル。選択したストリームのカラムと対応するデータ型が表示されます。 これは、参照情報としてのみ表示されます。このテーブルからパブリッシュするカラムは選択できません。 ただし、カラムの名前をコピーし、Excel に貼り付けることはできます。
Log File	ログ・ファイルのパスとファイル名を指定。このフィールドに直接入力するか、このフィールドの隣のボタンをクリックして、ファイル名とパスを参照し、選択します。 これはオプション・パラメータで、パブリッシュ時に発生したエラーが書き込まれるログ・ファイルの名前を指定します。 エラーは、このファイルに書き込まれるだけでなく、[Result] ボックスにも表示されます。
Result	パブリッシュの結果を表示する読み込み専用ボックス。
Publish Data	すべてのデータを入力したら、このボタンをクリックしてデータを Event Stream Processor にパブリッシュ。 パブリッシュの結果は、[Result] ボックスに表示されます。 データが Event Stream Processor にパブリッシュされると、Event Stream Processor ではデータが受信されたかどうかの確認だけが行われます。重複したレコード挿入や不正なデータなどの理由でレコードが拒否されたかどうかを確認するには、ストリームにサブスクライブするか、SQL クエリを発行する必要があります。

コンポーネント	説明
Show Formula	この画面のすべての情報を入力したら、式をグラフィカルに作成するためにこのボタンをクリック。 これによって、自動パブリッシュ用の式が容易に作成できます。 エラーがないと、式が [Result] ボックスに表示されます。 この式をコピーして Excel ワークシートに貼り付けることによって、データの自動パブリッシュを起動できます。
Clear Results	非常に多くのエントリがある場合に [Result] フィールドをクリアするには、このボタンをクリック。

自動パブリッシュ

SybaseRTP アドイン関数を使用して、セルが変更された場合には常にデータが自動的にパブリッシュされるようにします。

SybaseRTP は、基になる Excel リアルタイム・データ・メカニズムのラッパ関数で、データをパブリッシュするために使用されます。 この式の構文は次のとおりです。

```
=SybaseRTP("ConnectionName","StreamName","OperationCode",DataRange,
[[ColumnRange],[TransposeRows],["LogFile"],[InstanceNo],
[NoResults])]
```

以下に、各パラメータを説明します。

パラメータ	説明
`ConnectionName`	パブリッシュに使用する接続の名前。 接続を確立すると、正常にパブリッシュできるようになります。
`StreamName`	データのパブリッシュ先のストリームの名前。
`OperationCode`	パブリッシュ用に適用される必要のある opcode。 INSERT、UPDATE、DELETE、または UPSERT のいずれかを指定できます。
`DataRange`	パブリッシュするデータのある Excel 範囲のアドレスまたは名前。 DataRange オブジェクトは二重引用符で囲まないでください。
`[ColumnRange]`	(オプション) ストリーム・カラム名のある Excel の範囲アドレスまたは範囲名を指定。 このパラメータは引用符で囲まないでください。

パラメータ	説明
[TransposeRows]	(オプション) データ・レコードが、ローではなくカラムで指定されていることを指定。 true または false を指定できます。 デフォルト値は false です。
[LogFile]	(オプション) すべてのエラーがログ記録されるログ・ファイルの名前とロケーションを指定。 指定しないと、ロギングは一切行われません。
[InstanceNo]	内部で使用。 この値は、常に空のままにします。
[NoResults]	内部で使用。 この値は、常に False または空に設定します。

例を示します。

```
=SybaseRTP("Connection1","Trades","INSERT",A2:E10, A1:E1,False,"C:
¥logs¥log1.log",,)
```

上記の式は、ワークブック内のいずれのシートにも配置できます。 式を作成するときには、該当するワークシートの適切なセルを選択するか、[Workbook]Worksheet!A1:E5 式を使用して、アドレスがどのワークブックとワークシートを参照しているかを Excel に指示します。

式を Excel に配置すると、任意のセルに対して変更が行われるごとに、範囲全体がパブリッシュされます。 ただし、特定のセルが変更された場合のみパブリッシュされるようにするには、この関数をいつ呼び出すかを制御するビジネス・ロジックをカプセル化するカスタム・ラッパ内にこの呼び出しを配置します。

この関数の戻り値は配列で、Excel スタイルのロケーション表記 {{val11,val12},{val21,22}....} を使用する文字列としてフォーマットされます。 この式は、Excel スタイルの配列オブジェクトに変換できます。 この文字列には、要素の 1 つ以上の配列があり、各サブ要素には、2 つのサブ項目があります。 渡された値にエラーがあると、配列文字列には、1 つの要素のみがあります。 他の場合、パブリッシュするローの数よりも 1 つ多い要素があります。

配列文字列の最初の要素は、パブリッシュが完全に成功したかどうかを示すサマリです。 式の解析時にエラーが検出されると、

```
{{"1","Some error message."}}
```

形式の 1 要素配列が返されます。

レコードの検証時にエラーが検出されるか、パブリッシュが正常に完了すると、パブリッシュするローの数よりも 1 つ多い配列要素があります。 たとえば、2 つのローがパブリッシュされ、両方のレコードが正常にパブリッシュされると、配列文字列は、次の例のようになります。

```
{{"0",""},{"0",""},{"0",""}}
```

1 つのレコードのみが正常にパブリッシュされ、他方のレコードは失敗すると、次のような配列文字列が返されます。

```
{{"1","An error message"},{"0",""},{"1","row level error message"}}
```

サブスクリプション・クエリ

クエリに関連付けられた式のある Excel ワークシートを保存することによって、サブスクリプション・クエリを永続的に保存できます。

次回 Excel ワークシートを開いたときに、クエリが SybaseRT サブスクリプション作成ウィザードに表示され、正常に処理されます。

クエリの適用

[Subscription Wizard] ペインでクエリを適用して起動し、Excel ワークシートに値を設定します。

前提条件

[Subscription Wizard] ペインに、定義されたクエリがあること。

手順

1. [Subscription Queries] メニューから、該当するクエリを選択します。
2. [[適用]] をクリックします。
   - SybaseRT は最初に、提供されたサブスクリプション・クエリ名が既に使用されていないことを検証し、次に、提供された [Start Cell] の値が有効な Excel のセル・アドレスであることを検証します。 いずれかの検証が失敗した場合は、問題を解決する必要があります。
   - 次に、SybaseRT は、指定されたサブスクリプション・クエリに基づいて、Excel のリアルタイム・データ式を構築し、セルごとに 1 つの式をアクティブなワークシートに挿入します。 クエリによっては、数百の式挿入が実行されることがあります。 SybaseRT は以下のロジックを使用して式を挿入します。
     - 式は、常に、[Start Cell] で指定されたロケーションで始まるグリッドとして挿入される。 選択した各カラムは、分離しているが、連続しているカラムとして Excel ワークシートに表示されます。 [Max Rows] の値は、フィルタが適用されるローの数を制御します。
     - 最初の式がアクティブなワークシートに挿入されると、Excel はリアルタイム・データ式を認識し、最初のフィルタのクエリ情報を渡す呼び出しを、SybaseRT サーバに対して実行する。 リアルタイム・サーバは渡された情報を検査し、新しいクエリであると認識すると、クエリ・オブ

ジェクトを生成します。 リアルタイム・データ・サーバは、渡された情報を以降で使用するために格納します。
- このプロセスは、クエリの式ごとに繰り返される。ただし、リアルタイム・サーバは、その式を以前に認識したクエリの一部であると判断します。 このため、新しいクエリ・オブジェクトは作成されません。式に対応するデータを返せるようにするために、情報が格納されるだけです。

3. [Start] をクリックします。
   - SybaseRT は、Event Stream Processor への接続がアクティブで、指定されたクエリが引き続き有効であることを検証する。 これらの条件のいずれかが失敗すると、SybaseRT は戻します。
   - 次に、SybaseRT は新しい読み込みスレッドを生成してトランザクション・ログ・データを Event Stream Processor から読み込み、内部バッファに書き込む。
   - SybaseRT は 0.1 秒ごとにトランザクション・ログを内部バッファから読み込み、表示バッファ内でレコードの挿入、更新、または削除のいずれを実行するかを、ユーザ指定のキー・フィールドに基づいて決定する。 表示バッファへの挿入があり、バッファ内のレコード数が [Max Rows] で指定されている値に等しい場合、バッファ内の最も古いレコードが削除され、残りのレコードが 1 つずつ繰り上げられて、新しいレコードが末尾に挿入されます。 レコードの更新が必要な場合、置き換え更新が実行されます。この挿入と更新のメカニズムは、Excel ワークシートでの安定したデータ表示をもたらし、サブスクライブされたデータに基づくチャートの作成を容易にします。
   - 表示バッファにデータが格納されると、SybaseRT は Excel に対して、表示するための新しいデータがあることを通知する。 データ要求を受信すると、Excel が認識できる形式でデータを送信し、ワークシートの適切なロケーションに表示します。

既知の問題と制限事項

ESP Add-in for Microsoft Excel には、既知の問題と制限事項があります。

- [Max Rows] を、たとえば数千以上のローなど大きな値に設定すると、パフォーマンスが劣化することがある。 マシンが、要求を処理し、完了しようとして、非常にビジーになります。
- Event Stream Processor が停止するか、接続がネットワーク障害によって失われると、SybaseRT の画面が自動的にリフレッシュされず、クエリと接続の現在の状態が反映されなくなる。 接続またはクエリを選択して画面をリフレッシュし、選択したオブジェクトの現在の状態を表示します。
- Excel の同じインスタンス内では、SybaseRT の接続またはサブスクリプションの情報があるワークシートを複数使用できない。

FIX アダプタ

アダプタのタイプ： fixplugin。 Sybase Event Stream Processor FIX アダプタは、Sybase Event Stream Processor API と統合されたオープンソース QuickFIX エンジンの実装です。

FIX アダプタは、以下を実行します。

- 適切に動作する FIX エンジンを使用して FIX セッションを処理、管理する。
- コネクタと FIX セッションを経由して FIX メッセージを送受信する。
- インバウンド FIX メッセージを検証する。
- FIX メッセージを Event Stream Processor レコードに変換する。
- Event Stream Processor レコードを FIX メッセージに変換する。

注意： FIX アダプタは、FIX 辞書のカスタマイズをサポートします。

FIX アダプタは、Sybase 製品ダウンロード・サイトから取得できるライセンスを別途購入する必要があります。 標準の SySAM 猶予期間がサポートされます。つまり、ライセンスなしで 30 日間実行できます。 この期間が過ぎた場合、実行するには有効なライセンスが必要です。

参照：
- *FIX ファイル・インプット・アダプタ* (52 ページ)
- *FIX ファイル・アウトプット・アダプタ* (54 ページ)
- *ソケット FIX インプット・アダプタ* (96 ページ)
- *ソケット FIX アウトプット・アダプタ* (98 ページ)

サポート対象の FIX バージョン

FIX アダプタによってサポートされる FIX プロトコル・バージョン。

FIX アダプタは、FIX プロトコル・バージョン 4.0 ～ 5.0 をサポートします。

注意： FIXML はサポートされません。

制御フロー

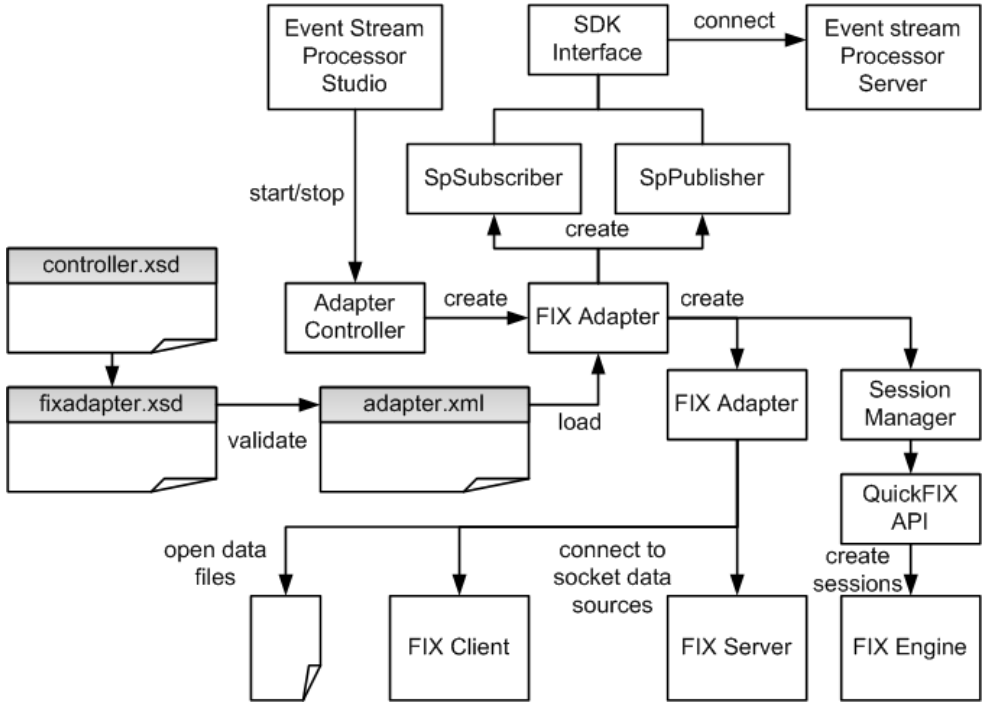
アダプタは、設定をファイルからロードし (たとえば、`adapter.xml`)、API 全体のコントローラ・スキーマ (`controller.xsd`) で構成されるアダプタ・スキーマ (`fixadapter.xsd`) に照らしてそのアダプタを検証します。

スキーマは編集できません。

FIX アダプタ制御フローには、異なる設定ファイル、さまざまなコマンドとコンポーネントのロードがあります。

アダプタ・コントローラは、アダプタのインスタンスを作成し、ユーザ・コマンドを受信、実行します。 アダプタ・コントローラは **start**、**stop**、**status** コマンドを実行します。

図 2 : 制御フロー



start コマンド

start コマンドは、FIX アダプタの起動、コマンドと制御のインタフェースの設定と起動、FIX 辞書、SpPublisher コンポーネント、SpSubscriber コンポーネントのロード、API インタフェースを介した Event Stream Processor への接続を実行します。

Message Distributor は、データ・ストリームのパブリッシュとサブスクライブを準備します。 データ・ストリームは、ストリーム・クラスタと呼ばれる階層に編成されます。 ストリーム・クラスタは、特定のタイプの FIX メッセージをホストできる一連のストリームです。

コネクタ・マネージャは、FIX データ・ファイル、FIX データのクライアントとサーバのデータソースへのソケット接続を開きます。また、セッション・マネージャは、QuickFIX API を使用して、上位互換性のある FIX エンジンでセッションを作成、ログオンします。 SpSubscriber と SpPublisher のコンポーネントは、API イ

ンタフェースを介して Event Stream Processor に接続します。 SpSubscriber は出力ストリームの受信を開始し、SpPublisher はデータを入力ストリームにパブリッシュする準備を整えます。

アダプタの実行中のインスタンスが存在するときに **start** コマンドを実行すると、このコマンドは無視され、警告が送信されます。

参照：
- *データ・ストリーム* (149 ページ)
- *メッセージ・フロー* (154 ページ)
- *FIX アダプタの起動* (178 ページ)

stop コマンド
stop コマンドによって、データソース・ハンドラがセッションを閉じ、データソースから切断します。また、アダプタ・コントローラはユーザ・コマンドの受信を停止し、アダプタ・プロセスは終了します。

コネクタ・マネージャは、開いているすべてのデータ・ファイル、クライアントとサーバのデータソースへのソケット接続を閉じ、セッション・マネージャは既存のセッションをログアウトします。

アダプタの実行中のインスタンスが存在しないときに **stop** コマンドを実行すると、このコマンドは無視され、警告が送信されます。

参照：
- *FIX アダプタの停止* (180 ページ)

status コマンド
status コマンドは、FIX アダプタのステータスを報告します。アダプタ・コントローラが次のステータスを表示します。 実行中または停止中のいずれか。

参照：
- *FIX アダプタのステータスの確認* (179 ページ)

データ・ストリーム

インプット FIX メッセージは、ストリーム・クラスタとして編成されるストリーム・レコードとして格納されます。

FIX アダプタは、個々のメッセージをストリーム・クラスタと呼ばれるストリーム階層に属している複数のレコードに格納します。 ストリーム・クラスタの最上位ストリームは、メイン・ストリームと呼ばれます。 メイン・ストリームは、FIX メッセージに属しているフィールドを格納します。ストリーム・クラスタの

その他のストリームはすべて、ネストされたグループに属しているフィールドを格納します。

注意： 同じタイプのメッセージは、複数のストリーム・クラスタに格納できます。 これらのクラスタは、共通の構造を共有する必要はありません。

インバウンド・メッセージをソース・ストリームのみに格納し、アウトバウンド・メッセージは、任意の種類のストリームに格納します。

FIX アダプタは、インバウンド・メッセージに関連したレコードのインデックスが適切に作成されていることを確認します。アウトバウンド・レコードのインデックスが適切に作成されているかどうかは、モデルを作成した担当者の責任になります。

アダプタの `templates` ディレクトリには、すべての FIX メッセージ・タイプの生成されたモデルがあります。 これらのモデルは自動的に生成され、特定のビジネス目的に役立つストリーム・クラスタを作成するために包括的なプロジェクトを使用できます。

参照：

- *メッセージ・フロー* (154 ページ)
- *start コマンド* (148 ページ)

例：FIX インプット・アダプタ・データ・ストリーム

FIX インプット・アダプタ・データ・ストリームのサンプル

これは `Quote` タイプの FIX メッセージです。

8=FIX.4.4 | 9=204 | 35=S | 49=COUNTERPARTYA | 55=AASymbol | 117=AAQuoteID | 133=31.1 | 453=2 | 448=AAPartyID1 | 447=B | 452=1 | 802=2 | 523=AAPartySubID11 | 803=1 | 523=AAPartySubID12 | 803=2 | 448=AAPartyID2 | 447=C | 452=2 | 802=1 | 523=AAPartySubID21 | 803=3 | 10=107 |

このデータ・ストリームには、次のフィールドがあります。

- SenderCompID=COUNTERPARTYA (tag 49)
- QuoteID=AAQuoteID (tag 117)
- Symbol=AASymbol (tag 55)
- OfferPx=31.1 (tag 133)
- NoPartyIDs=2 (tag 453)

Event Stream Processor のメッセージは、次のメイン・ストリームにあります。

```
<SourceStream id="MyQuotes" store="FixStore">
    <Column datatype="string" name="SenderCompID"/>
    <Column datatype="string" name="QuoteID"/>
    <Column datatype="integer" name="NoPartyIDs"/>
    <Column datatype="string" name="Symbol"/>
```

```
    <Column datatype="float" name="OfferPx"/>
    <Column datatype="long" name="FixMsgId" key="true"/>
</SourceStream>
```

次はサーバのメッセージです。

```
CREATE  MEMORY  STORE FixStore PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;


CREATE  INPUT  WINDOW MyQuotes
SCHEMA (SenderCompID STRING, QuoteID STRING, NoPartyIDs INTEGER,
Symbol STRING, OfferPx FLOAT, FixMsgId LONG)
PRIMARY KEY (FixMsgId)
 STORE FixStore;
```

メッセージには、2 つのグループの NoPartyID タイプがあります。

グループ 1：

- PartyID=AAPartyID1 (tag 448)
- PartyIDSource=B (tag 447)
- PartyRole=1 (Executing Firm, tag 452)
- NoPartySubIDs=2 (tag 802)

グループ 2：

- PartyID=AAPartyID1 (tag 448)
- PartyIDSource=C (tag 447)
- PartyRole=2 (Broker of Credit, tag 452)
- NoPartySubIDs=1 (tag 802)

Event Stream Processor のグループ 1 とグループ 2 は、次のストリームに格納されます。

```
<SourceStream id="MyQuotes_NoPartyIDs" store="FixStore">
    <Column datatype="string" name="PartyID"/>
    <Column datatype="string" name="PartyIDSource"/>
    <Column datatype="integer" name="PartyRole"/>
    <Column datatype="integer" name="NoPartySubIDs"/>
    <Column datatype="long" name="FixMsgId" key="true"/>
    <Column datatype="long" name="NoPartyIDs_Num" key="true"/>
</SourceStream>
```

サーバのグループ 1 とグループ 2 は、次のストリームに格納されます。

```
CREATE  INPUT  WINDOW MyQuotes_NoPartyIDs
SCHEMA (PartyID STRING, PartyIDSource STRING, PartyRole INTEGER,
NoPartySubIDs INTEGER, FixMsgId LONG, NoPartyIDs_Num LONG)
PRIMARY KEY (FixMsgId, NoPartyIDs_Num)
 STORE FixStore;
```

グループ 1 とグループ 2 には、独自のグループの NoPartySubID タイプがあります。 以下のグループ 11 とグループ 12 は、グループ 1 の一部です。

グループ 11：

- `PartySubID=AAPartySubID11 (tag 523)`
- `PartySubIDType=1 (Firm, tag 803)`

グループ 12：

- `PartySubID=AAPartySubID12 (tag 523)`
- `PartySubIDType=2 (Person, tag 803)`

グループ 21 はグループ 2 の一部です。

- `PartySubID=AAPartySubID21 (tag 523)`
- `PartySubIDType=3 (System, tag 803)`

Event Stream Processor のグループ 11、グループ 12、グループ 21 は、次のストリームに格納されます。

```
<SourceStream id="MyQuotes_NoPartyIDs_NoPartySubIDs"
store="FixStore">
    <Column datatype="string" name="PartySubID"/>
    <Column datatype="integer" name="PartySubIDType"/>
    <Column datatype="long" name="FixMsgId" key="true"/>
    <Column datatype="long" name="NoPartyIDs_Num" key="true"/>
    <Column datatype="long" name="NoPartySubIDs_Num" key="true"/>
<SourceStream>
```

サーバのグループ 11、グループ 12、グループ 21 は、次のストリームに格納されます。

```
CREATE  INPUT  WINDOW MyQuotes_NoPartyIDs_NoPartySubIDs
SCHEMA (PartySubID STRING, PartySubIDType INTEGER, FixMsgId LONG,
NoPartyIDs_Num LONG, NoPartySubIDs_Num LONG)
PRIMARY KEY (FixMsgId, NoPartyIDs_Num, NoPartySubIDs_Num)
 STORE FixStore;
```

ストリームとカラム名

ストリーム・カラムのフィールド名が FIX フィールドに対応していることを確認します。 カラムの順序は、FIX 辞書のフィールドの順序に従う必要はありません。

注意： ホストされた FIX メッセージと無関係なカラムは使用できません。

メイン・ストリームの名前は、任意に選択できます。

下位ストリームが命名規則に厳密に従っていることを確認します。 それぞれの下位ストリームには親ストリームがあり、繰り返しグループに対応しているため、その名前が次の形式に従っていることを確認します。

```
<parent stream name>_<name of the repeating group>
```

ヘッダ・フィールドとトレーラ・フィールド

ヘッダ・フィールドとトレーラ・フィールドを追加または更新して、有効な FIX メッセージを作成できます。 一部のカラムは、ヘッダ・フィールドまたはトレー

ラ・フィールドに対応している場合があります。 出力コネクタは、メッセージ本文のすべてのフィールドを変更しないで、ストリーム・カラムに格納されたまま維持します。

レコードのインデックスの作成

FIX メッセージは、レコードの階層に格納され、インデックス・カラムを使用して相互参照されます。

インデックス・カラムには、`long` タイプがあり、ストリームの最後に位置しています。

メイン・ストリームのレコードには、1 つのインデックスのみがあります。 子レコードには 2 つのインデックスがあり、1 つ目は親レコードと同じ値を持ちます。 次のレベルの下位レコードには 3 つのインデックスがあり、最初の 2 つは親レコードと同じ値を持つ必要があり、以下のレベルでも同様になります。

アダプタとセッション

FIX アダプタは、他の FIX エンジンと同様、FIX メッセージをファイルやソケット接続などのデータソースと交換します。

ファイルとソケット接続は、コネクション・マネージャによって処理されます。 他の FIX エンジンを使用するセッションは、セッション・マネージャによって処理されます。

注意： このアダプタは、いくつでも種類の異なるデータソースを同時に処理できます。

アダプタは一方向です。 各アダプタを使用して、ファイル、クライアント・ソケット接続、サーバ・ソケット接続など、FIX データの 1 つのソースからメッセージを受信 (またはそれらのソースに送信) できます。 デフォルトで、ファイルとソケットのアダプタは、すべての入力メッセージのチェックサムと本体長タグを検証します。 アダプタ設定ファイルで検証をオフにできます。

他の FIX エンジンを使用するセッションは、双方向です。 セッションを介して受信する入力メッセージは、必ず検証されます。セッションの検証はオフにできません。 セッション管理 (ログイン、ログアウト、メッセージ・シーケンス番号、メッセージの再送など) とメッセージ検証は、QuickFIX API によって実行されます。

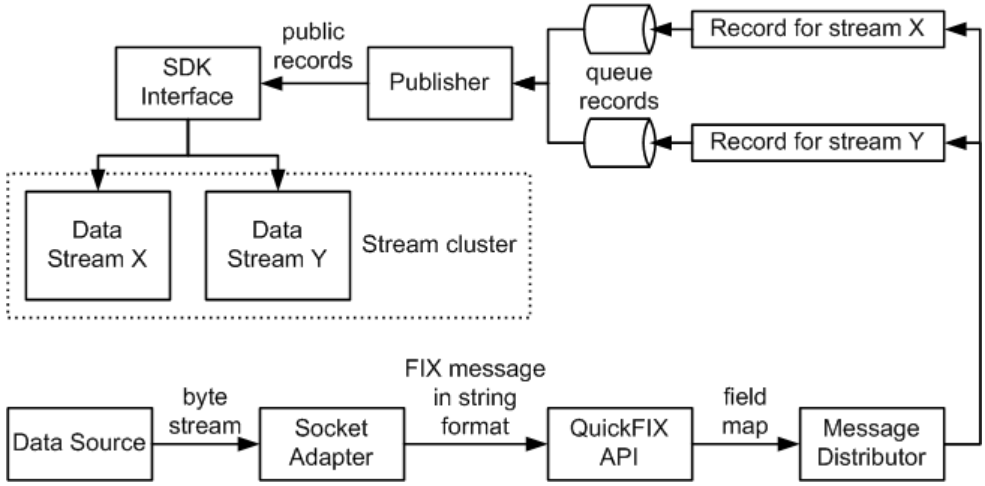
無効な入力メッセージは破棄され、すべてのエラーがログ記録されます。 それ以外の場合、チェックサムや本体長タグがない、または正しくない値がある場合でも、メッセージは解析され、ストリーム・クラスタにパブリッシュされます。

アダプタは、出力メッセージを送信する前に、チェックサムと body length を再計算し、適切なタグを更新します。

メッセージ・フロー

アダプタを流れるメッセージ・フローは、**start** コマンドによって開始されます。

図 3 : ソケット・コネクタを流れるインバウンド・メッセージ・フロー



コネクタは FIX データをバイト・ストリームとして受信します。 FIX メッセージは、文字列オブジェクトに事前解析されます。 QuickFIX API は、その文字列を解析してフィールド・マップに格納し、それらのフィールド・マップは Message Distributor に渡されます。

Message Distributor は、各フィールド・マップをストリーム・クラスタに合わせて複数のレコードに変換します。 これで、レコードは Event Stream Processor にパブリッシュするための準備が整いました。 ただし、それらのレコードはすぐにはパブリッシュされません。 レコードはキューに入れられてから、個別のスレッド (それぞれのレコード・キューに対して 1 つのスレッド) 上のパブリッシャ・オブジェクトによって取り出されます。 キュー容量は設定できます。 キューのサイズを大きくすると、突発的に大量のメッセージが発生してもオーバフローが起こりにくくなります。 キューに登録されているレコードがキューの容量の 4 分の 3 を超えると、警告がログ記録されます。容量が 4 分の 3 未満に戻ると、別の警告がログ記録されます。 キューが満杯になると、アダプタは空きが生じるまで待機してから、次のレコードを配置します。

レコードは非同期にパブリッシュされます。 アダプタは、いかなるフィードバックも Event Stream Processor から受信しません。

Event Stream Processor でアダプタを使用していて、障害が発生した場合、SDK のインタフェースは、メッセージを失うことなく、予備の Event Stream Processor インスタンスに切り替えます。

参照：
- *データ・ストリーム* (149 ページ)
- *start コマンド* (148 ページ)

FIX アダプタのデータ型のマッピング
Event Stream Processor のデータ型は、FIX データ型にマップされます。

Event Stream Processor のデータ型	QuickFix のデータ型
`integer`	`boolean`
`string`	`byte[]`
`string`	`char`
`string`	`string`
`date`	`date`
`float`	`float`
`integer`	`integer`
`date` または `timestamp`	`UTCDateOnly`
`date` または `timestamp`	`UTCTimeOnly`
`date` または `timestamp`	`UTCTimeStamp`

JAVA_HOME 環境変数の設定
Java ディレクトリを指すように JAVA_HOME 環境変数を設定します。

前提条件
Java Runtime Environment のバージョン 1.6.0_26 以降をインストールする。

手順
JAVA_HOME 環境変数を、Java Runtime Environment 1.6.0_26 以降がインストールされているディレクトリ・パスに設定します。

次のステップ
ESP_HOME 環境変数が正しく設定されていることを確認する。

設定

FIX アダプタに関する設定情報。

FIX アダプタのディレクトリ

アダプタのディレクトリには、設定ファイル、テンプレート、例、JAR ファイルなど、アダプタに関連するすべてのファイルがあります。

```
README.txt    Documentation note
ReleaseNotes.txt        Release notes

bin/
    adapter.bat        Standalone adapter startup script
    adapter.sh        Standalone adapter startup script
    adapter-plugin.bat        Plug-in connector startup script
 adapter-plugin.sh        Plug-in connector startup script

config/
    controller.xsd        Controller schema
    fixadapter.xsd        Adapter schema
    log4j.properties    Sample logging configuration
    login.config        Authentication configuration

dictionary/
    FIX40.xml        FIX 4.0 dictionary
    FIX41.xml        FIX 4.1 dictionary
    FIX42.xml        FIX 4.2 dictionary
    FIX43.xml        FIX 4.3 dictionary
    FIX44.xml        FIX 4.4 dictionary

examples/ Working examples
    AllInOne/
    ClientSocketConnectors/
    FileConnectors/
    ServerSocketConnectors/

lib/
    quickfixj/        QuickFIX libraries
        mina-core-1.1.0.jar
        quickfixj-all-1.3.2.jar
        slf4j-api-1.5.6.jar
        slf4j-simple-1.5.6.jar
  sylapij.dll (Windows)
  libsylapij.so (UNIX)
    esp_fix_adapter.jar FIX    Adapter code
templates/        Platform templates
    FlexStream40.xml
    FlexStream41.xml
    FlexStream42.xml
    FlexStream43.xml
    FlexStream44.xml
    SourceStream40.xml
    SourceStream41.xml
    SourceStream42.xml
```

```
    SourceStream43.xml
    SourceStream44.xml
```

スキーマと設定ファイル
アダプタ設定はファイルからロードされ、アダプタのスキーマに対して検証されます。

FIX アダプタ設定ファイルが `$ESP_HOME/adapters/fix/config` に配置されていることを確認してから、アダプタを起動します。また、アダプタ設定がスキーマに対して有効であることも確認します。

`$ESP_HOME/adapters/fix/examples` フォルダには、サンプル・アダプタ設定ファイルがあります。 これらのファイルのいずれかの編集、または新規ファイルの記述もできます。

注意： アダプタ・マネージャは、設定ファイルで `<sp>` ノードまたは `<sdk>` ノードのいずれかを探します。`<sp>` ノードは Event Stream Processor への接続を示します。

アダプタ制御パラメータ
アダプタ・コントローラ・ポートは、コマンドを受信します。

パラメータ名	タイプ	説明
controllerPort	`positive integer`	(必須) アダプタの「コマンドと制御」のポートを指定。 ユーザ・コマンドは、localhost 上のこのポートに送信されます。 **注意：** それぞれのアダプタ・インスタンスに、専用ポートがあることを確認します。

Event Stream Processor のパラメータ
Event Stream Processor パラメータは、Event Stream Processor と FIX アダプタの間の通信を設定します。

これらのパラメータは、`controller.xsd` ファイル (`config` ディレクトリ内) で定義されます。

パラメータ名	タイプ	説明
espAuthType	`string`	(必須) Event Stream Processor への認証に使用される方法を指定。 有効な値は、以下のとおりです。 • **server_rsa –** キーストアを使用する RSA 認証 • **user_password –** Kerberos 認証と LDAP 認証 • **none –** 認証なし アダプタがスタジオ・プラグインとして動作している場合、**espAuthType** は **Authentication Mode** スタジオ起動パラメータで上書きされます。
espUser	`string`	(必須) Event Stream Processor にログインするために必要なユーザ名を指定。 none (**espAuthType** を参照してください) 以外のすべての認証スキームで必要です。 デフォルト値はありません。
espPassword	`string`	(必須) Event Stream Processor にログインするために必要なパスワードを指定。 none (**espAuthType** を参照してください) 以外のすべての認証スキームで必要です。 **espPassword** 値が暗号化されているかどうかを示す "encrypted" 属性も指定します。デフォルト値は false です。 true に設定すると、パスワード値は **espRSAKeyStore** と **espRSAKeyStorePassword** を使用して復号化されます。
espProjectUri	`string`	(必須) Event Stream Processor クラスタに接続するための完全なプロジェクト URI を指定。たとえば、`esp://localhost:19011/ws1/p1` のように記述します。

パラメータ名	タイプ	説明
pulseInterval	`non-negative integer`	(必須) 期間を秒単位で指定。この期間では、アウトバウンド・レコードの変更は Event Stream Processor によって結合され、アダプタによって単一のイベントとして受信されるようになります。 設定されていないか、0 に設定されると、レコードの変更は、発生するごとに個々に受信されます。
espHeartbeatPeriod	`positive integer`	(オプション) 秒数を指定。この秒数が経過すると、アダプタは次のハートビートを Event Stream Processor に送信します。 Event Stream Processor が連続して 2 回のハートビート受信に失敗すると、アダプタがパブリッシュしたすべてのレコードが失効とマーク付けされます。 デフォルト値は 10 です。
recordQueueCapacity	`positive integer`	(オプション) レコード・キューの容量を指定。デフォルト値は、4096 です。
maxPubPoolSize	`positive integer`	(オプション) レコード・プールの最大サイズを指定。 レコード・プーリングは、パブリッシュ処理の高速化に役立ちます。デフォルト値は 256 です。
maxPubPoolTime	`positive integer`	(オプション) レコードがパブリッシュされるまでにプールに存在可能な最大時間をミリ秒単位で指定。 設定されていない場合、プーリング時間は無制限で、プーリング方式は **maxPubPoolSize** によって制御されます。 デフォルト値はありません。
useTransactions	`boolean`	(オプション) true に設定すると、プールされたメッセージがトランザクションで Event Stream Processor にパブリッシュされる。 false に設定すると、エンベロープでパブリッシュされます。デフォルト値は false です。

パラメータ名	タイプ	説明
espRSAKeyStore	`string`	(場合に応じて必須) RSA キーストアのロケーションを指定し、パスワード値を復号化するために使用される。 **espAuthType** が server_rsa に設定されている、**espPassword** の暗号化属性が true に設定されている、または両方の場合に必要です。
espRSAKeyStorePassword	`string`	(場合に応じて必須) キーストア・パスワードを指定し、パスワード値を復号化するのに使用される。**espAuthType** が server_rsa に設定されている、**espPassword** の暗号化属性が true に設定されている、または両方の場合に必要です。
espEncryptionAlgorithm	`string`	(オプション) **espPassword** の暗号化属性が true に設定されると使用される。 ブランクのままにすると、RSA がデフォルトとして使用されます。

FIX インプット・アダプタ

FIX インプット・アダプタは、任意の数のファイル、ソケット、セッション・コネクタから FIX メッセージを読み込みます。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Adapter Directory Path	**baseDir**	`directory`	(必須) アダプタ・ベース・ディレクトリへのパスを指定する。 デフォルト値はありません。 **注意：** このプロパティは、[Connector Remote Directory Path] プロパティが指定されていると無視されます。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Configuration File Path	**configFilePath**	`configFilename`	(必須) アダプタ設定ファイルへの絶対パスを指定する。 デフォルト値はありません。 **注意：** このプロパティは、[Remote Configuration File Path] プロパティが指定されていると無視されます。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。
Adapter Remote Directory	**remoteBaseDir**	`string`	(詳細) アダプタ・リモート・ベース・ディレクトリへのパスを指定 (リモート実行のみ)。 デフォルト値はありません。 **注意：** このプロパティが指定されている場合、[Connector Directory Path] プロパティは無視されます。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Adapter Configuration File Path	**remoteConfigFilePath**	`string`	(詳細) アダプタ設定ファイルへの絶対パスを指定 (リモート実行のみ)。 デフォルト値はありません。 **注意：** このプロパティが指定されている場合、[Configuration File Path] プロパティは無視されます。

Event Stream Processor サーバ・プロパティ

サーバ接続プロパティは、設定ファイルの `<sdk>` ノードの属性とサブ要素として提供されます。

パラメータ名	説明
server	(必須) **sdk** ノードの属性を指定します。
serverHost	(必須) サーバが稼働しているマシンの名前を指定します。
serverPort	(必須) サーバが接続を受信するポート番号を指定します。
serverWorkspace	(必須) ワークスペース。
serverProject	(必須) プログラム名。
serverUser	(オプション) ユーザ名。
serverPassword	(オプション) パスワード。

FIX 辞書

FIX アダプタの辞書には、FIX メッセージ・タイプ、コンポーネント、フィールドの定義があります。

パラメータ名	タイプ	説明
fixDictionary	`string`	(必須) FIX 辞書ファイルの名前。 **注意：** すべてのサポートされる FIX バージョン (4.0 ～ 5.0) の辞書ファイルは、`dictionary` フォルダにあります。 FIX メッセージ・タイプ、コンポーネント、フィールドの定義を編集できます。

ストリーム設定

設定ファイルの `streams` セクションを使用して、FIX メッセージ・タイプをストリーム・クラスタにマップします。

パラメータ名	タイプ	説明
name	`string`	(必須) ストリーム・クラスタ内のメイン・ストリームの名前。 デフォルト値はありません。
messageName	`string`	(必須) ストリーム・クラスタでホストされるメッセージ・タイプの名前。 デフォルト値はありません。

`MyQuotes` ストリームの下流のストリーム・クラスタでタイプ `Quote` の FIX メッセージをホストするには、このフラグメントを `<streams>` グループに追加します。

```
<stream>
    <name>MyQuotes</name>
    <messageName>Quote</messageName>
</stream>
```

コネクタ

FIX 設定ファイルの `connector` セクションは、ファイル・コネクタとソケット・コネクタを定義します。

パラメータ名	タイプ	説明
streamNames	`streamNameType`	(必須) このコネクタを通るメッセージがホストされるストリーム・クラスタのメイン・ストリームの名前を示す。

インバウンド・コネクタとアウトバウンド・コネクタ
FIX 設定ファイルの **inbound** パラメータと **outbound** パラメータは、インバウンド・ファイル・コネクタ、アウトバウンド・ファイル・コネクタ、インバウンド・ソケット・コネクタ、アウトバウンド・ソケット・コネクタをリストします。

パラメータ名	タイプ	説明
doValidation	`boolean`	(オプション) true に設定すると、このコネクタを通るインバウンド・メッセージは、メッセージ長とチェックサムが正しいことが検証される。 false に設定すると、メッセージ長フィールドとチェックサム・フィールドは無視されます。 無効なメッセージは破棄され、エラーがログ記録されます。 デフォルト値は true です。 **注意：** メッセージ・データは、メッセージの解析中にデータ辞書に対して検証されます。

参照：
- *例： オール・イン・ワンの使用* (187 ページ)

オール・イン・ワン・コネクタ用のサンプル設定ファイル
FIX アダプタのオール・イン・ワン・コネクタ用サンプル設定ファイル (`adapter.xml`)。

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!--  Sybase ESP FIX adapter configuration file
  -->
- <adapter>
- <!--  Adapter Controller
  -->
- <controller>
  <controllerPort>13579</controllerPort>
  </controller>
- <!--  Event Stream Processor settings
  -->
- <esp>
- <espConnection>
  <espProjectUri>esp://localhost:19011/w1/p1</espProjectUri>
  </espConnection>
- <espSecurity>
  <espUser>espuser</espUser>
  <espPassword encrypted="false">espuser</espPassword>
  <espAuthType>none</espAuthType>
- <!--         <espRSAKeyStore>/keystore/keystore.jks</
espRSAKeyStore>
        <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword>
  -->
  <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
```

```
  </espSecurity>
  </esp>
- <!--  FIX dictionary
  -->
  <fixDictionary>FIX44.xml</fixDictionary>
- <!--  Stream cluster to FIX message mapping
  -->
- <streams>
- <stream>
  <name>MyQuotes</name>
  <messageName>Quote</messageName>
  </stream>
- <stream>
  <name>MyOrders</name>
  <messageName>NewOrderSingle</messageName>
  </stream>
  </streams>
- <!--  Connectors
  -->
- <connectors>
- <outbound>
- <fileConnector>
  <fileName>orders.fix</fileName>
- <streamNames>
  <streamName>MyOrders</streamName>
  </streamNames>
  </fileConnector>
  </outbound>
  </connectors>
- <!--  FIX Session Settings
  -->
- <sessionSettings>
- <default>
  <ConnectionType>acceptor</ConnectionType>
  <SocketAcceptPort>23456</SocketAcceptPort>
  <FileLogPath>logs</FileLogPath>
  <FileStorePath>store</FileStorePath>
  <DataDictionary>FIX44.xml</DataDictionary>
  <HeartBtInt>600</HeartBtInt>
  <BeginString>FIX.4.4</BeginString>
  <StartTime>00:00:00</StartTime>
  <EndTime>23:59:59</EndTime>
  <SenderCompID>SYBASE</SenderCompID>
  </default>
- <sessionSetting>
  <TargetCompID>COUNTERPARTYA</TargetCompID>
  </sessionSetting>
- <sessionSetting>
  <TargetCompID>COUNTERPARTYB</TargetCompID>
  </sessionSetting>
  </sessionSettings>
- <!--  Session logins
  -->
- <sessionLogins>
- <senderLogin>
  <username>MyUsername</username>
```

```
  <password>MyPassword</password>
  <NextExpectedMsgSeqNum>1</NextExpectedMsgSeqNum>
  </senderLogin>
- <targetLogin>
  <TargetCompID>COUNTERPARTYA</TargetCompID>
  <username>UsernameA</username>
  <password>PasswordA</password>
  </targetLogin>
- <targetLogin>
  <TargetCompID>COUNTERPARTYB</TargetCompID>
  <username>UsernameB</username>
  <password>PasswordB</password>
  </targetLogin>
  </sessionLogins>
- <!--  Sessions
  -->
- <sessions>
- <inbound>
- <session>
  <TargetCompID>COUNTERPARTYA</TargetCompID>
- <streamNames>
  <streamName>MyQuotes</streamName>
  </streamNames>
  </session>
  </inbound>
- <outbound>
- <session>
  <TargetCompID>COUNTERPARTYA</TargetCompID>
- <streamNames>
  <streamName>MyOrders</streamName>
  </streamNames>
  </session>
- <session>
  <TargetCompID>COUNTERPARTYB</TargetCompID>
- <streamNames>
  <streamName>MyQuotes</streamName>
  </streamNames>
  </session>
  </outbound>
  </sessions>
  </adapter>
```

ファイル・コネクタ

FIX 設定ファイルの **fileConnector** パラメータは、ファイル・コネクタのプロパティ値をリストします。

パラメータ名	タイプ	説明
fileName	`string`	(必須) FIX データが構成されているファイルへの相対パスまたは絶対パス。

パラメータ名	タイプ	説明
streamNames	`streamNamesType`	(必須) このコネクタを通るメッセージがホストされるストリーム・クラスタのメイン・ストリームの名前を示す。
doValidation	`boolean`	(オプション) true に設定すると、このコネクタを通るインバウンド・メッセージは、メッセージ長とチェックサムが正しいことが検証される。 false に設定すると、メッセージ長フィールドとチェックサム・フィールドは無視されます。 無効なメッセージは破棄され、エラーがログ記録されます。 デフォルト値は true です。 **注意：** メッセージ・データは、メッセージの解析中にデータ辞書に対して検証されます。

参照：
- *例： ファイル・コネクタの使用* (181 ページ)

ファイル・コネクタ用のサンプル設定ファイル
FIX アダプタのファイル・コネクタ用サンプル設定ファイル (`adapter.xml`)。

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!--  Sybase ESP FIX adapter configuration file
  -->
- <adapter>
- <!--  Adapter Controller
  -->
- <controller>
  <controllerPort>13579</controllerPort>
  </controller>
- <!--  Event Stream Processor settings
  -->
- <esp>
- <espConnection>
  <espProjectUri>esp://localhost:19011/w1/p1</espProjectUri>
  </espConnection>
- <espSecurity>
  <espUser>espuser</espUser>
  <espPassword encrypted="false">espuser</espPassword>
  <espAuthType>none</espAuthType>
- <!--         <espRSAKeyStore>/keystore/keystore.jks</
espRSAKeyStore>
        <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword>
  -->
  <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
  </espSecurity>
  </esp>
- <!--  FIX dictionary
```

```
  -->
  <fixDictionary>FIX44.xml</fixDictionary>
- <!--  Stream cluster to FIX message mapping
  -->
- <streams>
- <stream>
  <name>MyQuotes</name>
  <messageName>Quote</messageName>
  </stream>
- <stream>
  <name>MyOrders</name>
  <messageName>NewOrderSingle</messageName>
  </stream>
  </streams>
- <!--  Connectors
  -->
- <connectors>
- <inbound>
- <fileConnector>
  <fileName>quotes.fix</fileName>
  <doValidation>false</doValidation>
- <streamNames>
  <streamName>MyQuotes</streamName>
  </streamNames>
  </fileConnector>
  </inbound>
- <outbound>
- <fileConnector>
  <fileName>orders.fix</fileName>
- <streamNames>
  <streamName>MyOrders</streamName>
  </streamNames>
  </fileConnector>
  </outbound>
  </connectors>
- <!--  Sessions
  -->
  <sessions />
  </adapter>
```

クライアント・ソケット・コネクタ

FIX 設定ファイルの **clientSocketConnector** パラメータは、FIX メッセージの送信先のサーバの名前、IP アドレス、検証スキーム、ストリーム名、ポートを定義します。

パラメータ名	タイプ	説明
dataHost	`string`	(必須) FIX メッセージの送信先のサーバの名前または IP アドレスを指定。
dataPort	`nonNegativeInteger`	(必須) 接続先のポートを指定。

パラメータ名	タイプ	説明
streamNames	`streamNameType`	(必須) このコネクタを通るメッセージがホストされるストリーム・クラスタのメイン・ストリームの名前を示す。
doValidation	`boolean`	(オプション) true に設定すると、このコネクタを通るインバウンド・メッセージは、メッセージ長とチェックサムが正しいことが検証される。 false に設定すると、メッセージ長フィールドとチェックサム・フィールドは無視されます。 無効なメッセージは破棄され、エラーがログ記録されます。 デフォルト値は true です。 **注意：** メッセージ・データは、メッセージの解析中にデータ辞書に対して検証されます。

参照：
- *例： クライアント・ソケット・コネクタの使用* (182 ページ)

クライアント・ソケット・コネクタ用のサンプル設定ファイル
FIX アダプタのクライアント・ソケット・コネクタ用サンプル設定ファイル (`adapter.xml`)。

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <!--  Sybase ESP FIX adapter configuration file
  -->
- <adapter>
- <!--  Adapter Controller
  -->
- <controller>
  <controllerPort>13579</controllerPort>
  </controller>
- <!--  Event Stream Processor settings
  -->
- <esp>
- <espConnection>
  <espProjectUri>esp://localhost:19011/w1/p1</espProjectUri>
  </espConnection>
- <espSecurity>
  <espUser>espuser</espUser>
  <espPassword encrypted="false">espuser</espPassword>
  <espAuthType>none</espAuthType>
- <!--         <espRSAKeyStore>/keystore/keystore.jks</
espRSAKeyStore>
        <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword>
  -->
```

```
  <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
  </espSecurity>
  </esp>
- <!--  FIX dictionary
  -->
  <fixDictionary>FIX44.xml</fixDictionary>
- <!--  Stream cluster to FIX message mapping
  -->
- <streams>
- <stream>
  <name>MyQuotes</name>
  <messageName>Quote</messageName>
  </stream>
- <stream>
  <name>MyOrders</name>
  <messageName>NewOrderSingle</messageName>
  </stream>
  </streams>
- <!--  Connectors
  -->
- <connectors>
- <inbound>
- <clientSocketConnector>
  <dataHost>localhost</dataHost>
  <dataPort>43210</dataPort>
  <doValidation>true</doValidation>
- <streamNames>
  <streamName>MyQuotes</streamName>
  </streamNames>
  </clientSocketConnector>
  </inbound>
- <outbound>
- <clientSocketConnector>
  <dataHost>localhost</dataHost>
  <dataPort>32109</dataPort>
- <streamNames>
  <streamName>MyOrders</streamName>
  </streamNames>
  </clientSocketConnector>
  </outbound>
  </connectors>
- <!--  Sessions
  -->
  <sessions />
  </adapter>
```

サーバ・ソケット・コネクタ
FIX 設定ファイルの **serverSocketConnector** パラメータは、サーバがクライアント接続を受信するポートを定義します。

パラメータ名	タイプ	説明
dataPort	`nonNegativeInteger`	(必須) サーバがクライアント接続を受信するポートを指定。
streamNames	`streamNameType`	(必須) このコネクタを通るメッセージがホストされるストリーム・クラスタのメイン・ストリームの名前を示す。
doValidation	`boolean`	(オプション) true に設定すると、このコネクタを通るインバウンド・メッセージは、メッセージ長とチェックサムが正しいことが検証される。 false に設定すると、メッセージ長フィールドとチェックサム・フィールドは無視されます。 無効なメッセージは破棄され、エラーがログ記録されます。 デフォルト値は true です。 **注意：** メッセージ・データは、メッセージの解析中にデータ辞書に対して検証されます。

参照：
- *例： サーバ・ソケット・コネクタの使用* (184 ページ)

サーバ・ソケット・コネクタ用のサンプル設定ファイル
FIX アダプタのサーバ・ソケット・コネクタ用サンプル設定ファイル (`adapter.xml`)。

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!--  Sybase ESP FIX adapter configuration file
  -->
- <adapter>
- <!--  Adapter Controller
  -->
- <controller>
  <controllerPort>13579</controllerPort>
  </controller>
- <!--  Event Stream Processor settings
  -->
- <esp>
```

```
- <espConnection>
  <espProjectUri>esp://localhost:19011/w1/p1</espProjectUri>
  </espConnection>
- <espSecurity>
  <espUser>espuser</espUser>
  <espPassword encrypted="false">espuser</espPassword>
  <espAuthType>none</espAuthType>
- <!--         <espRSAKeyStore>/keystore/keystore.jks</
espRSAKeyStore>
        <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword>
  -->
  <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
  </espSecurity>
  </esp>
- <!--  FIX dictionary
  -->
  <fixDictionary>FIX44.xml</fixDictionary>
- <!--  Stream cluster to FIX message mapping
  -->
- <streams>
- <stream>
  <name>MyQuotes</name>
  <messageName>Quote</messageName>
  </stream>
- <stream>
  <name>MyOrders</name>
  <messageName>NewOrderSingle</messageName>
  </stream>
  </streams>
- <!--  Connectors
  -->
- <connectors>
- <inbound>
- <serverSocketConnector>
  <dataPort>54321</dataPort>
  <doValidation>true</doValidation>
- <streamNames>
  <streamName>MyQuotes</streamName>
  </streamNames>
  </serverSocketConnector>
  </inbound>
- <outbound>
- <serverSocketConnector>
  <dataPort>43210</dataPort>
- <streamNames>
  <streamName>MyOrders</streamName>
  </streamNames>
  </serverSocketConnector>
  </outbound>
  </connectors>
- <!--  Sessions
  -->
  <sessions />
  </adapter>
```

セッション設定
`sessionSettings` セクションのプロパティは、FIX エンジンとのセッション接続のデフォルトと特定の設定を設定します。

デフォルト設定
すべてのセッション接続のデフォルト設定を設定するために使用されるプロパティ。

プロパティ名	タイプ	説明
ConnectionType	`string`	(必須) アダプタがサーバまたはクライアントのどちらとして動作するかを指定。有効な値は以下のとおりです。 • **Acceptor –** アダプタは、FIX セッション・イニシエータから接続要求を受け付けるサーバとして動作する。 • **Initiator –** アダプタは、FIX セッション・アクセプタに接続するクライアントとして動作する。 デフォルト値はありません。 **注意：** 各アダプタ・インスタンスは、アクセプタ・モードまたはイニシエータ・モードのどちらかとして動作し、同時に両方のモードで動作できません。
SocketAcceptPort	`nonNegativeInteger`	(必須) アダプタが FIX セッション・イニシエータからの接続を受信するポートを指定。 デフォルト値はありません。 イニシエータ・モードでのみ動作します。
FileLogPath	`string`	(必須) メッセージ・ログのディレクトリ・パスを指定。 絶対パスも相対パスも受け入れられます。 デフォルト値はありません。
FileStorePath	`string`	(必須) メッセージ・ストアのディレクトリ・パスを指定。 絶対パスも相対パスも受け入れられます。 デフォルト値はありません。

プロパティ名	タイプ	説明
StartTime	`string`	(必須) FIX セッションをアクティブにする時刻を指定。 デフォルト値はありません。
EndTime	`string`	(必須) FIX セッションを非アクティブにする時刻を指定。 デフォルト値はありません。
DataDictionary	`string`	(必須) FIX 辞書ファイル・パスへの絶対パスまたは相対パスを指定。 デフォルト値はありません。
BeginString	`string`	(必須) アウトバウンド FIX メッセージの **BeginString** フィールド (タグ 8) の値を指定。 デフォルト値はありません。
SenderCompID	`string`	(必須) アウトバウンド FIX メッセージの **SenderCompID** フィールド (タグ 49) の値を指定。 アダプタ・インスタンスは、すべてのセッション接続に対して、同じ **SenderCompID** 値を使用します。
HeartBtInt	`nonNegativeInteger`	(オプション) ハートビート間隔を秒単位で指定。 デフォルト値は 10 です。 イニシエータ・モードでのみ動作します。
ReconnectInterval	`positiveInteger`	(オプション) 再接続試行の間隔を秒単位で指定。 アダプタはスタートアップ時にアクセプタ・エンジンへの接続に失敗した場合、またはその後に接続が失われると、再接続を試行し続けます。 デフォルト値は 30 です。 イニシエータ・モードでのみ動作します。

プロパティ名	タイプ	説明
LogonTimeout	`nonNegativeInteger`	(オプション) ログオン応答を待機する時間を秒単位で指定。この時間が経過すると、アクセプタ・エンジンとの接続を切断します。 デフォルト値は 10 です。 イニシエータ・モードでのみ動作します。
LogoutTimeout	`nonNegativeInteger`	(オプション) ログアウト応答を待機する時間を秒単位で指定。この時間が経過すると、アクセプタ・エンジンとの接続を切断します。 デフォルト値は 2 です。 イニシエータ・モードでのみ動作します。

特定の設定
特定のセッション接続の特定の設定を設定するために使用されるパラメータ。

プロパティ名	タイプ	説明
TargetCompID	`string`	(必須) アウトバウンド FIX メッセージの **TargetCompID** フィールド (タグ 56) の値を指定。
SocketConnectHost	`string`	(必須) アクセプタ・エンジンのホスト名または IP アドレスを指定。 **注意：** イニシエータ・モードでのみ動作します。
SocketConnectPort	`non-negative integer`	(必須) アクセプタ・エンジンが接続を受信するポートを指定。 **注意：** イニシエータ・モードでのみ動作します。

セッション・ログイン
`sessionLogins` セクションのプロパティは、FIX エンジンとのセッション接続の送信側とターゲットのログイン・プロパティを設定します。

送信側のログイン・プロパティ
指定されると、アダプタは、FIX セッションの開始時に、**senderLogin** プロパティをアウトバウンド・ログイン・メッセージに付加します。

パラメータ名	タイプ	説明
username	`string`	(必須) 送信側のユーザ名を指定。
password	`string`	(必須) 送信側のパスワードを指定。
NextExpectedMsgSeqNum	`integer`	(必須) アウトバウンド・ログイン・メッセージの **NextExpectedMsgSeqNum** フィールド (タグ 789) の値を指定。

ターゲットのログイン・プロパティ
各インバウンド・ログイン・メッセージに対して、FIX アダプタは、**username** フィールドと **password** フィールドの値が、対応する **TargetCompID** フィールドで指定されているものと一致しているかどうかを検証します。

注意： ユーザ名またはパスワードが一致しない場合、エラーがログ記録されます。

パラメータ名	タイプ	説明
username	`string`	(必須) ターゲットのユーザ名を指定。
password	`string`	(必須) ターゲットのパスワードを指定。
TargetCompID	`string`	(必須) インバウンド・ログイン・メッセージの **TargetCompID** フィールド (タグ 56) の値を指定。

セッション・プロパティ
sessionProperties セクションのプロパティは、インバウンドとアウトバウンドの FIX セッションを識別し、FIX セッションで交換されるデータをホストするストリーム・クラスタのメイン・ストリームを示します。

プロパティ名	タイプ	説明
TargetCompID	`string`	(必須) セッション識別子。 デフォルト値はありません。 **注意：** 同じ識別子のインバウンド・セッションとアウトバウンド・セッションは、単一の双方向セッションとして実装されます。

プロパティ名	タイプ	説明
streamNames	`streamNamesType`	(必須) この FIX セッション上で交換されるメッセージがホストされるストリーム・クラスタのメイン・ストリームの名前のリストを指定。 • インバウンド・セッションの場合、マップされていないメッセージは無視され、マップされているメッセージは、マップ先のすべてのストリーム・クラスタに書き込まれる。 • アウトバウンド・セッションの場合、ヘッダ・フィールドまたはトレーラ・フィールドが必要に応じて追加または更新され、送信 FIX メッセージの有効性が維持される。 **注意：** 同じタイプのメッセージをホストする 2 つのストリーム・クラスタが同じ構造である必要はありません。

例： インバウンド・メッセージの受信とホスティング
指定されたターゲットからのインバウンド・メッセージを受信し、指定されたメイン・ストリームが存在する指定されたストリーム・クラスタでホストします。

`COUNTERPARTYA` として識別されるターゲットから FIX セッションを介して受信されるインバウンド・メッセージは、`MyQuotes` がメイン・ストリームであるストリーム・クラスタでホストされます。 相場以外のタイプのメッセージは無視されます。

```
<inbound>
    <session>
        <TargetCompID>COUNTERPARTYA</TargetCompID>
        <streamNames>
            <streamName>MyQuotes</streamName>
        </streamNames>
    </session>
</inbound>
```

ロギング
FIX アダプタは Apache `log4j` API を使用して、エラー、警告、情報、デバッグ・メッセージをログ記録します。

ロギング設定が構成されているサンプル `log4j.properties` ファイルは、FIX アダプタ配布の一部として提供されます。 このファイルは、`$ESP_HOME/adapters/fix/examples/<example name>/log4j.properties` フォルダにあります。

注意： ログ記録レベルを `DEBUG` に設定すると、ログ・ファイルが非常に大きくなることがあります。デフォルト値は `INFO` です。

ログ記録設定の詳細については、http://logging.apache.org/log4j を参照してください。

オペレーション

FIX アダプタをコマンド・ラインから操作します。

`adapter.xml` 設定を任意のロケーションに配置できるようにするために、**start**、**stop**、**status** のコマンドでファイルのフル・パスを指定します。

注意： 環境変数として、長いファイル・パス名を定義できます。

FIX アダプタの起動

FIX アダプタをコマンド・ラインから起動するには、Event Stream Processor を起動し、パラメータを検証して、**start** コマンドを実行します。

1. Event Stream Processor を起動します。

   Windows：

   1. サンプル・クラスタを起動します。

      ```
      cd %ESP_HOME%¥cluster¥nodes¥node1
        %ESP_HOME%¥bin¥esp_server --cluster-node node1.xml
      ```

   2. CCL をコンパイルして CCX を作成します。

      ```
      %ESP_HOME%¥bin¥esp_compiler -i model.ccl -o model.ccx
      ```

   3. プロジェクトをクラスタに展開します。

      ```
      %ESP_HOME%¥bin¥esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --add_project --workspace-
      name=w1 --project-name=p1 --ccx=model.ccx
      ```

   4. 展開したクラスタをサーバで起動します。

      ```
      %ESP_HOME%¥bin¥esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --start_project --
      workspace-name=w1 --project-name=p1
      ```

   UNIX：

   1. サンプル・クラスタを起動します。

      ```
      cd $ESP_HOME/cluster/nodes/node1
        $ESP_HOME/bin/esp_server --cluster-node node1.xml
      ```

   2. CCL をコンパイルして CCX を作成します。

      ```
      $ESP_HOME/bin/esp_compiler -i model.ccl -o model.ccx
      ```

   3. プロジェクトをクラスタに展開します。

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --add_project --workspace-
name=w1 --project-name=p1 --ccx=model.ccx
```

4. 展開したクラスタをサーバで起動します。

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --start_project --
workspace-name=w1 --project-name=p1
```

2. アダプタを起動します。

オペレーティング・システム	手順

UNIX	端末ウィンドウを開き、次を入力。
	`cd $ESP_HOME/adapters/fix/bin`
	`./adapter.sh <configuration file path> start`

Windows	コマンド・ウィンドウを開き、次を入力。
	`cd %ESP_HOME%/adapters/fix/bin`
	`adapter.bat <configuration file path> start`

esp_subscribe ユーティリティを使用して、FIX メッセージが正しく Event Stream Processor にパブリッシュされることを確認できます。

参照：

- *start コマンド* (148 ページ)

FIX アダプタのステータスの確認

status コマンドを端末ウィンドウまたはコマンド・ウィンドウから実行して、アダプタのステータスを確認します。

FIX アダプタのステータスの確認

オペレーティング・システム	手順

UNIX	端末ウィンドウを開き、次を入力。
	`cd $ESP_HOME/adapters/fix/bin`
	`./adapter.sh <configuration file path> status`

オペレーティング・システム	手順

Windows	コマンド・ウィンドウを開き、次を入力。
	`cd %ESP_HOME%/adapters/fix/bin`
	`adapter.bat <configuration file path> status`

アダプタのステータスが、実行中または停止中のいずれかとして表示されます。

参照：
- *status コマンド* (149 ページ)

FIX アダプタの停止
stop コマンドを端末ウィンドウまたはコマンド・ウィンドウから実行して、アダプタを停止します。

オペレーティング・システム	手順

UNIX	端末ウィンドウを開き、次を入力。
	`cd $ESP_HOME/adapters/fix/bin`
	`./adapter.sh $ADAPTER stop &`

Windows	コマンド・ウィンドウを開き、次を入力。
	`cd %ESP_HOME%/adapters/fix/bin`
	`adapter.bat %ADAPTER% stop`

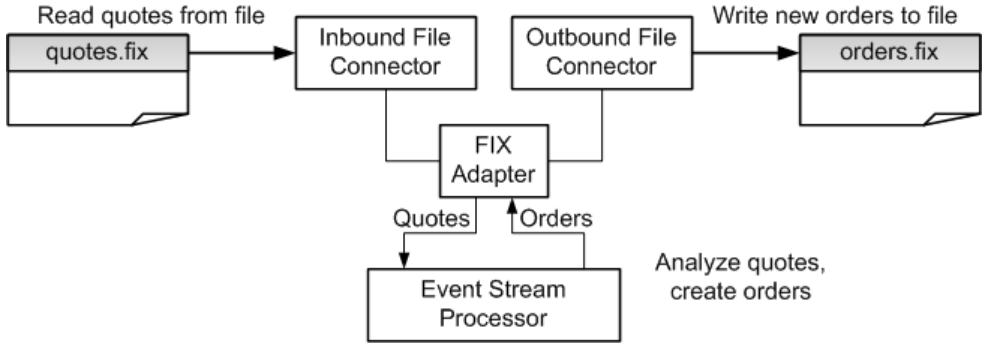
参照：
- *stop コマンド* (149 ページ)

例
アダプタと共に提供される、実際に動作する例を使用して、リアルタイム・マーケット・データにサブスクライブし、データを Event Stream Processor にパブリッシュする方法を学習します。

例で提供されているスクリプトは、ネットワーク接続を必要としません。

例： ファイル・コネクタの使用
ファイル・コネクタを使用してファイルから相場メッセージを読み込み、Event Stream Processor にパブリッシュします。 OfferPx フィールドの値が 30.0 未満の場合、NewOrderSingle メッセージを別のファイルに書き込みます。

図 4 : ファイル・コネクタ



1. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`
Windows	コマンド・ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.bat` 2. プロジェクトをクラスタに追加し、クラスタ上で起動。 `start_project.bat`

2. Event Stream Processor のそれぞれのサブスクライバ・ユーティリティを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./esp-subscribe.sh`

オペレーティング・システム	手順
Windows	コマンド・ウィンドウを開き、次を入力。 `esp-subscribe.bat`

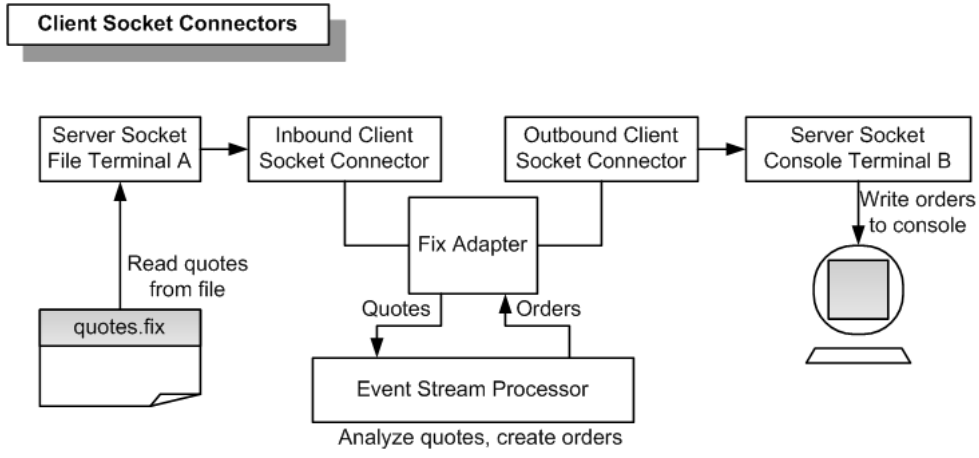
3. アダプタを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./adapter.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `adapter.bat`

4. アダプタの初期化が完了するまで 5 ～ 10 秒待機します。
   - この時点で、MyQuotes ストリームには 2 つのレコードがある。 MyOrders ストリームには、1 つのレコードがあります。 ストリームの内容を表示するには、プラットフォーム・サブスクライバ・ユーティリティを使用します。
   - この時点で、`orders.fix` ファイルには、1 つの NewOrderSingle メッセージがあります。

参照：
- *ファイル・コネクタ* (166 ページ)

例： クライアント・ソケット・コネクタの使用
クライアント・ソケット・コネクタを使用して相場メッセージをサーバ・ソケットから読み込み、プラットフォームにパブリッシュします。 OfferPx フィールドの値が 30.0 未満の場合、アダプタは NewOrderSingle メッセージを別のサーバ・ソケットに書き込みます。



1. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`

Windows	コマンド・ウィンドウを開く。
	1. サンプル・クラスタを起動。
	`start_server_cluster.bat`
	2. プロジェクトをクラスタに追加し、クラスタ上で起動。
	`start_project.bat`

2. Event Stream Processor のそれぞれのサブスクライバ・ユーティリティを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./esp-subscribe.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `esp-subscribe.bat`

3. アダプタを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./adapter.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `adapter.bat`

4. アダプタの初期化が完了するまで 5 ～ 10 秒待機します。
5. サーバ・ソケット端末 B を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./terminalB.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `terminalB.bat`

6. サーバ・ソケット端末 A を起動します。

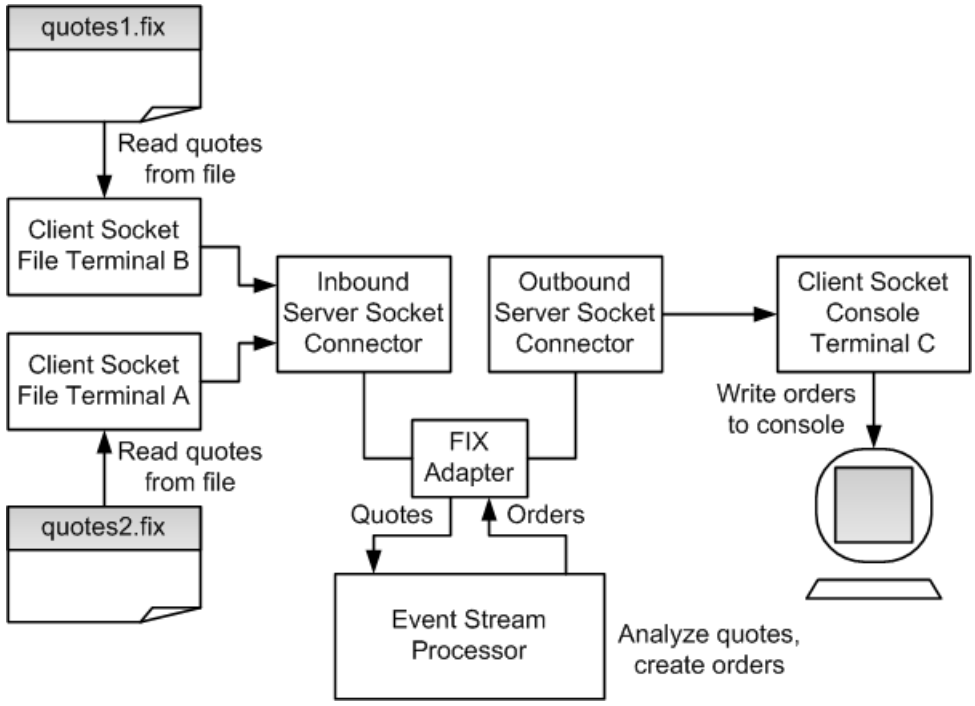
オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./terminalA.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `terminalA.bat`

- この時点で、MyQuotes ストリームには 2 つのレコードがあり、MyOrders ストリームには 1 つのレコードがある。 ストリームの内容を表示するには、プラットフォーム・サブスクライバ・ユーティリティを使用します。
- 端末 B コンソール・ウィンドウには、NewOrderSingle メッセージが表示される。

参照：
- *クライアント・ソケット・コネクタ* (168 ページ)

例： サーバ・ソケット・コネクタの使用
サーバ・ソケット・コネクタを使用して相場メッセージを 2 つのクライアント・ソケットから読み込み、プラットフォームにパブリッシュします。 OfferPx フィールドの値が 30.0 未満の場合、FIX アダプタは NewOrderSingle メッセージを 3 番目のクライアント・ソケットに書き込みます。

図 5 : サーバ・ソケット・コネクタ



1. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`
Windows	コマンド・ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.bat` 2. プロジェクトをクラスタに追加し、クラスタ上で起動。 `start_project.bat`

2. Event Stream Processor のそれぞれのサブスクライバ・ユーティリティを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./esp-subscribe.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `esp-subscribe.bat`

3. アダプタを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./adapter.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `adapter.bat`

4. アダプタの初期化が完了するまで 5 ～ 10 秒待機します。
5. 出力端末 C を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./terminalC.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `terminalC.bat`

6. 出力端末 B を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./terminalB.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `terminalB.bat`

7. 出力端末 A を起動します。

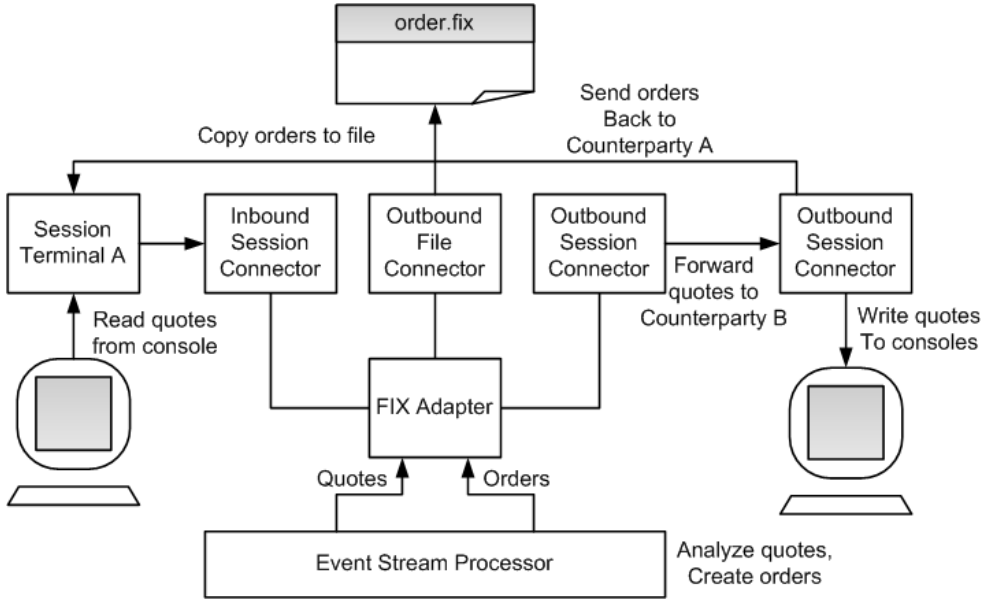
オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./terminalA.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `terminalA.bat`

- この時点で、MyQuotes ストリームには 3 つのレコードがある。 MyOrders ストリームには、1 つのレコードがあります。 ストリームの内容を表示するには、プラットフォーム・サブスクライバ・ユーティリティを使用します。
- 端末 C コンソール・ウィンドウには、NewOrderSingle メッセージが表示される。

参照：
- *サーバ・ソケット・コネクタ* (171 ページ)

例： オール・イン・ワンの使用
通信相手 A との双方向の FIX セッションと、通信相手 B との一方向の FIX セッションを使用します。OfferPx フィールドの値が 30.0 未満の場合、FIX アダプタは NewOrderSingle メッセージを通信相手 A に返し、ファイルにコピーします。

図 6 : オール・イン・ワン



1. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`

Windows	コマンド・ウィンドウを開く。
	1. サンプル・クラスタを起動。
	`start_server_cluster.bat`
	2. プロジェクトをクラスタに追加し、クラスタ上で起動。
	`start_project.bat`

2. Event Stream Processor のそれぞれのサブスクライバ・ユーティリティを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./esp-subscribe.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `esp-subscribe.bat`

3. アダプタを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./adapter.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `adapter.bat`

4. アダプタの初期化が完了するまで 5 ～ 10 秒待機します。
5. 出力端末 B を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./terminalB.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `terminalB.bat`

6. アダプタの初期化が完了するまで 5 ～ 10 秒待機します。
7. 出力端末 A を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./terminalA.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `terminalA.bat`

8. アダプタの初期化が完了するまで 5 ～ 10 秒待機します。
9. `quotes.fix` ファイルの内容をコピーし、端末 A のコンソール・ウィンドウに貼り付けます。

- この時点で、MyQuotes ストリームには 2 つのレコードがある。 MyOrders ストリームには、1 つのレコードがあります。 ストリームの内容を表示するには、プラットフォーム・サブスクライバ・ユーティリティを使用します。
- 端末 B コンソール・ウィンドウには、NewOrderSingle メッセージが表示される。
- この時点で、`orders.fix` ファイルには、NewOrderSingle メッセージのコピーがある。

参照：
- *インバウンド・コネクタとアウトバウンド・コネクタ* (164 ページ)

フレックス・アダプタ

Sybase Event Stream Processor のフレックス・アウトプット・アダプタはソフトウェア・インタフェースで、これを使用することによって、Event Stream Processor プロジェクトのストリームからデータを取得し、それらをすべての範囲の Adobe Flex アプリケーションに提供できます。

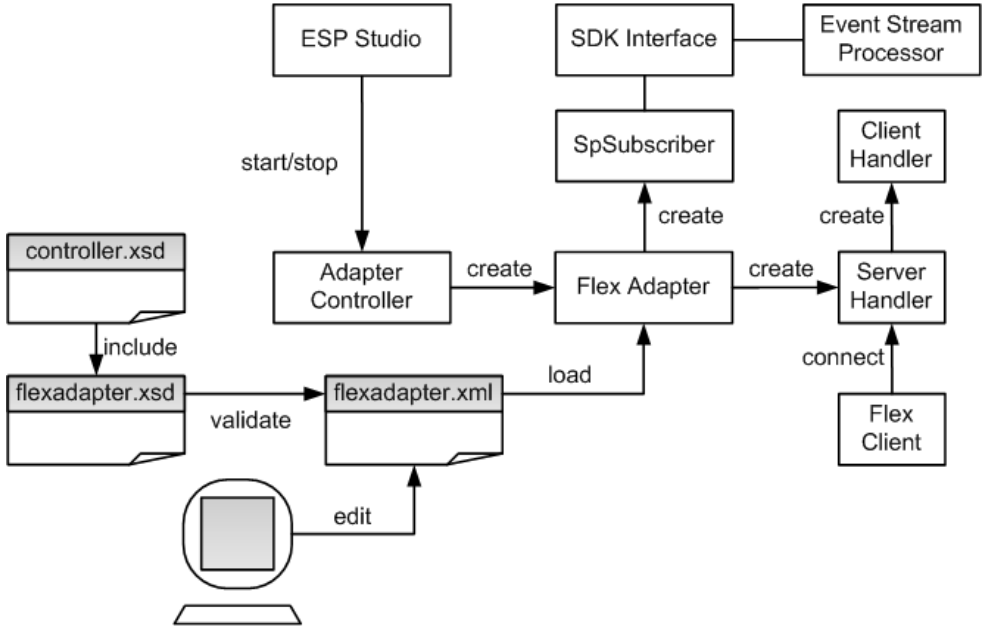
フレックス・アダプタは以下の機能を提供します。

- 内部フレックス・サーバを実行し、クライアント接続を受信して受け付ける。
- クライアント要求を解析し、ストリームにサブスクライブする。
- ストリーム・レコードをフィルタし、XML に変換して、1 回に 1 レコードずつクライアントに送信する。

制御フロー

フレックス・アダプタは設定をファイル (たとえば、`adapter-pubsub.xml`) からロードし、アダプタ・スキーマ (`flexadapter.xsd`) に対して検証します。このスキーマは、API 全体のコントローラ・スキーマ (`controller.xsd`) で構成されます。

図 7 : フレックス・アダプタ制御フロー



アダプタ・コントローラはアダプタのインスタンスを作成し、**start**、**stop**、**status** のコマンドを受け取り、実行します。

start コマンド

start コマンドは、制御インタフェースを設定して起動します。このコマンドの実行によって、Server Handler がクライアント接続の受信を開始し、各クライアント接続を処理するために個別の Client Handler が作成されます。また、SpSubscriber コンポーネントが Event Stream Processor に SDK インタフェースを介して接続されます。

アダプタの実行中のインスタンスが存在するときに **start** コマンドを実行すると、このコマンドは無視され、警告が送信されます。

参照：
- *フレックス・アダプタの起動* (200 ページ)

stop コマンド
stop コマンドは、SpSubscriber を Event Stream Processor から切断します。これによって、Server Handler は新しいクライアント接続の受け付けを停止し、Client Handler は既存のクライアントへの応答を終了し、クライアントを切断します。さらに、アダプタ・プロセスが終了します。

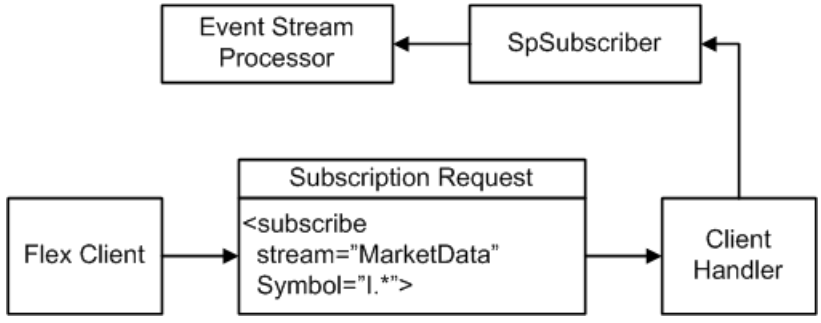
アダプタの実行中のインスタンスが存在しないときに **stop** コマンドを実行すると、コマンドは無視され、警告が送信されます。

参照：
- *フレックス・アダプタの停止* (202 ページ)

status コマンド
status コマンドは、アダプタのステータスを報告します。アダプタ・コントローラが次のステータスを表示します。 実行中または停止中のいずれか。

参照：
- *フレックス・アダプタのステータスの確認* (201 ページ)

メッセージ・フロー
アダプタと任意のフレックス・クライアントとの間のメッセージ・フローは、クライアントがストリーム名とカラム・フィルタ (オプション) を示すサブスクライブ要求を送信したときに開始されます。



要求が次のフォーマットであることを確認します。

```
<subscribe stream="MyStream" myFilterColumn1="MyRegex1" .../>
```

フィルタされたカラムが文字列型であることを確認します。 正規表現がカラム値として受け入れられます。 たとえば、次の要求が処理されると、

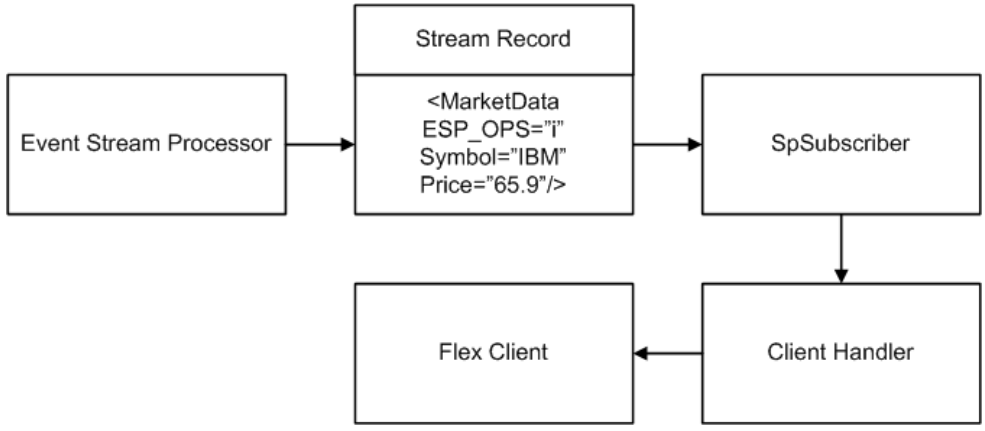
```
<subscribe stream="MarketData" Symbol="I.*"/>
```

クライアントは、Symbol が大文字の "I" で始まる、MarketData ストリームのすべてのレコードを受信します。

クライアントに送り戻されるレコードのフォーマットは、次のとおりです。

```
<MyStream ESP_OPS="i" myColumn1="MyValue1" myColumn2="MyValue2" .../
>
```

ここで、ESP_OPS はレコードの opcode です。 有効な opcode 値は、"i" (INSERT)、"u" (INSERT)、"d" (UPDATE)、"p" (UPSERT) です。 レコードには、opcode に関係なく、null 値以外のすべてのカラムがあります。 null のカラム値は無視されます。



以前にサブスクライブしたストリームからサブスクリプションを削除するには、クライアントは次のフォーマットの要求を送信します。

```
<unsubscribe stream="MyStream"/>
```

クライアントは、任意の数の異なるストリームに同時にサブスクライブできます。 カラム・フィルタ値を変更するには、最初のストリームからサブスクリプションを削除し、次に新しいフィルタを使用してサブスクライブします。

Event Stream Processor のフェールオーバが発生すると、SDK API はメッセージを失うことなく、パブリッシュ先を予備の Event Stream Processor インスタンスに切り替えます。

Stream Handler

クライアントサーバ通信用に Stream Handler を使用します。

フレックス・クライアントはロー XML を使用してサブスクライブし、Event Stream Processor からストリーム・データを受信できますが、Sybase はクライアントサーバ通信を Stream Handler に委任することをおすすめします。 Stream Handler の機能を使用するには、`lib` にある `streamhandler.swc` ライブラリをフレックス・クライアント・ビルドに追加します。

Stream Handler を ActionScript コードで使用する例を次に示します。

```
    import com.sybase.esp.adapter.flex.StreamHandler;
private var streamHandler:StreamHandler = new StreamHandler(
"localhost", 23456, onConnect, onDisconnect, onRecord);
private function onConnect(event:Event):void {
// Invoked after the client has successfully connected to
// the adapter
}
private function onDisconnect(event:Event):void {
// Invoked after the client has disconnected from
// the adapter
}
private function onRecord(streamName:String, opcode:String,
record:Object):void {
// Invoked when the client receives a record from
// the adapter
}
```

ストリームにサブスクライブするには、ストリーム名とフィルタ・パラメータを指定して、Stream Handler の `subscribe()` メソッドを呼び出します。 フィルタは ActionScript オブジェクトで、フィルタされるカラムごとに 1 つずつの複数のプロパティがあります。 プロパティ値として正規表現を使用できます。例を示します。

```
var filter:Object = new Object();
filter.Symbol = "I.*";
streamHandler.subscribe("Stream1", filter);
```

フィルタされたカラムが文字列型であることを確認します。 すべてのストリーム・レコードをフィルタリングなしで受け取るには、オブジェクトをフィルタするプロパティを何も追加しません。 ストリームからサブスクリプションを削除するには、ストリーム名とフィルタ・パラメータを指定して、Stream Handler の `unsubscribe()` メソッドを呼び出します。

例を示します。

```
streamHandler.subscribe("Stream1");
```

`onRecord()` コールバック・メソッドを実装して、サブスクリプションに到着するレコードを処理します。 コールバックでは、次の 3 つのパラメータを使用します。

- **streamName**
- **opcode**
- プロパティとしてすべての null 以外のカラム値があるオブジェクト

例を示します。

```
private function onRecord(streamName:String, opcode:String,
record:Object):void {
trace("Record received on stream " + streamName);
trace("Opcode=" + opcode);
trace("Column values:");
for (var column:String in record)
trace(column + "=" + record[column];
```

JAVA_HOME 環境変数の設定

Java ディレクトリを指すように JAVA_HOME 環境変数を設定します。

前提条件

- Java Runtime Environment のバージョン 1.6.0_26 以降をインストールする。 適切なバージョンの Java がインストールされていることを確認するには、http://www.java.com/en/download/installed.jsp?detect=jre"&"try=1 を参照してください。
- Java をダウンロードし、インストールするには、http://jdl.sun.com/webapps/getjava/BrowserRedirect?locale=en"&"host=www.java.com:80 にアクセスしてください。

手順

JAVA_HOME 環境変数を、Java Runtime Environment 1.6.0_26 以降がインストールされているディレクトリ・パスに設定します。

次のステップ

ESP_HOME 環境変数が正しく設定されていることを確認する。

設定

フレックス・アダプタ用の設定情報を以下に示します。

フレックス・アダプタのディレクトリ

アダプタのディレクトリには、設定ファイル、テンプレート、例、JAR ファイルなど、アダプタに関連するすべてのファイルがあります。

```
README.txt  Quick Guide
ReleaseNotes.txt  Release Notes
```

```
bin/
    adapter.bat  Standalone adapter startup script
    adapter.sh      Standalone adapter startup script
    adapter-plugin.bat Plug-in connector startup script
    adapter-plugin.sh  Plug-in connector startup script

config/
    controller.xsd     Controller schema
    log4j.properties   Sample logging configuration
    flexadapter.xsd    Flex Adapter schema
    login.config  Authentication configuration


example/        Working example

lib/
    esp_flex_adapter.jar  Flex adapter library

javadoc/
    adapterapi/     Adapter API Javadoc
    flexadapter/     Flex Adapter Javadoc


Common jars are located here:

$ESP_HOME/adapters/jar

The directory structure is:

activation.jar               Java mail library
adapterapi.jar               Adapter API code
axis.jar                    Webservices jar
commons-codec-1.3.jar        Required by SDK API
commons-discovery-0.2.jar
commons-logging-1.1.jar    Logging library
esp_java_sdk-0.2.jar     ESP SDK library
jaxrpc-api-1.1.jar               Required by ESP SDK
log4j-1.2.14.jar         Logging library
mail.jar              Java mail library
saaj-api-1.3.jar             Webservices jar
ws-commons-util-1.0.2.jar  Required by ESP SDK
wsdl4j-1.5.1.jar
xercesImpl.jar        XML parser library
xmlrpc-client-3.1.3.jar Required by ESP SDK
xmlrpc-common-3.1.3.jar Required by ESP SDK
xmlrpc-server-3.1.3.jar Required by ESP SDK
```

スキーマと設定ファイル

アダプタ設定はファイルからロードされ、アダプタのスキーマに対して検証されます。

例として提供されているフォルダには、サンプルのアダプタ設定ファイル `adapter.xml` があります。

有効な設定ファイルを提供し、アダプタ設定がアダプタ・スキーマに対して正しく検証されるようにする必要があります。

アダプタ・コントローラ・パラメータ

アダプタ・コントローラ・パラメータは、アダプタの「コマンドと制御」のポートを指定します。

このパラメータは、`config` ディレクトリの `controller.xsd` ファイル内で定義されます。

パラメータ名	タイプ	説明
controllerPort	`positive integer`	(必須) アダプタの「コマンドと制御」のポートを指定。 ユーザ・コマンドは、localhost 上のこのポートに送信されます。

Event Stream Processor のパラメータ

Event Stream Processor のパラメータは、Event Stream Processor とフレックス・アダプタとの間の通信を設定します。

これらのパラメータは、`controller.xsd` ファイル (`config` ディレクトリ内) で定義されます。

パラメータ名	タイプ	説明
espAuthType	`string`	(必須) Event Stream Processor への認証に使用される方法を指定。 有効な値は、以下のとおりです。 • **server_rsa –** キーストアを使用する RSA 認証 • **user_password –** Kerberos 認証と LDAP 認証 • **none –** 認証なし アダプタがスタジオ・プラグインとして動作している場合、**espAuthType** は **Authentication Mode** スタジオ起動パラメータで上書きされます。
espUser	`string`	(必須) Event Stream Processor にログインするために必要なユーザ名を指定。 none (**espAuthType** を参照してください) 以外のすべての認証スキームで必要です。 デフォルト値はありません。

パラメータ名	タイプ	説明
espPassword	`string`	(必須) Event Stream Processor にログインするために必要なパスワードを指定。 none (**espAuthType** を参照してください) 以外のすべての認証スキームで必要です。 **espPassword** 値が暗号化されているかどうかを示す "encrypted" 属性も指定します。デフォルト値は false です。 true に設定すると、パスワード値は **espRSAKeyStore** と **espRSAKeyStorePassword** を使用して復号化されます。
espProjectUri	`string`	(必須) Event Stream Processor クラスタに接続するための完全なプロジェクト URI を指定。たとえば、`esp://localhost:19011/ws1/p1` のように記述します。
pulseInterval	`non-negative integer`	(必須) 期間を秒単位で指定。この期間では、アウトバウンド・レコードの変更は Event Stream Processor によって結合され、アダプタによって単一のイベントとして受信されるようになります。 設定されていないか、0 に設定されると、レコードの変更は、発生するごとに個々に受信されます。
espHeartbeatPeriod	`positive integer`	(オプション) 秒数を指定。この秒数が経過すると、アダプタは次のハートビートを Event Stream Processor に送信します。 Event Stream Processor が連続して 2 回のハートビート受信に失敗すると、アダプタがパブリッシュしたすべてのレコードが失効とマーク付けされます。 デフォルト値は 10 です。
recordQueueCapacity	`positive integer`	(オプション) レコード・キューの容量を指定。デフォルト値は、4096 です。
maxPubPoolSize	`positive integer`	(オプション) レコード・プールの最大サイズを指定。 レコード・プーリングは、パブリッシュ処理の高速化に役立ちます。デフォルト値は 256 です。

パラメータ名	タイプ	説明
maxPubPoolTime	`positive integer`	(オプション) レコードがパブリッシュされるまでにプールに存在可能な最大時間をミリ秒単位で指定。 設定されていない場合、プーリング時間は無制限で、プーリング方式は **maxPubPoolSize** によって制御されます。 デフォルト値はありません。
useTransactions	`boolean`	(オプション) true に設定すると、プールされたメッセージがトランザクションで Event Stream Processor にパブリッシュされる。 false に設定すると、エンベロープでパブリッシュされます。デフォルト値は false です。
espRSAKeyStore	`string`	(場合に応じて必須) RSA キーストアのロケーションを指定し、パスワード値を復号化するために使用される。 **espAuthType** が server_rsa に設定されている、**espPassword** の暗号化属性が true に設定されている、または両方の場合に必要です。
espRSAKeyStorePassword	`string`	(場合に応じて必須) キーストア・パスワードを指定し、パスワード値を復号化するのに使用される。**espAuthType** が server_rsa に設定されている、**espPassword** の暗号化属性が true に設定されている、または両方の場合に必要です。
espEncryptionAlgorithm	`string`	(オプション) **espPassword** の暗号化属性が true に設定されると使用される。 ブランクのままにすると、RSA がデフォルトとして使用されます。

フレックス・サーバの設定

serverPort パラメータは、アダプタがフレックス・サーバを実行するポートを指定します。

このパラメータは、`config` ディレクトリの `flexadapter.xsd` ファイル内で定義されます。

パラメータ名	タイプ	説明
serverPort	`int`	(必須) アダプタがフレックス・サーバを実行するポートを指定。

フレックス設定ファイルのサンプル

フレックス・アダプタ用のサンプル設定ファイル (`adapter.xml`) を以下に示します。

このファイルは、`example` にあります。

```
<adapter>

<!-- Adapter Controller -->
 <controller>
  <controllerPort>13579</controllerPort>
 </controller>

<!-- Sybase ESP Server settings -->
 <esp>
   <espConnection>
        <espProjectUri>esp://localhost:19011/w1/p1</espProjectUri>
   </espConnection>

   <espSecurity>
        <espUser>espuser</espUser>
        <espPassword encrypted="false">espuser</espPassword>
        <espAuthType>none</espAuthType>
<!--  <espRSAKeyStore>/keystore/keystore.jks</espRSAKeyStore>
       <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword> --
>
        <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
   </espSecurity>
  <maxPubPoolSize>1</maxPubPoolSize>
 </esp>

<!-- Flex specific -->
 <flex>
  <serverPort>23456</serverPort>
 </flex>

</adapter>
```

ロギング

アダプタは Apache `log4j` API を使用して、エラー、警告、情報、デバッグ・メッセージをログ記録します。

`log4j.properties` ファイルには、ログ記録設定があります。 サンプル `log4j.properties` ファイルが、アダプタ配布と共に提供されます。

ログ記録レベルを DEBUG に設定すると、ログ・ファイルが非常に大きくなることがあります。 デフォルトのレベルは、INFO です。

ログ記録設定の詳細については、http://logging.apache.org/log4j を参照してください。

オペレーション

コマンド・ラインからアダプタを起動、停止、またはステータスの確認を実行します。

フレックス・アダプタの起動

フレックス・アダプタをコマンド・ラインから起動するには、Event Stream Processor を起動して、**start** コマンドを実行します。

前提条件

アダプタがクライアント接続を受信するポートが、フレックス・クライアントが実行されるマシンからの TCP 接続に対して開いていることを確認します。

手順

1. Event Stream Processor を起動します。

   Windows：

   1. サンプル・クラスタを起動します。

      ```
      cd %ESP_HOME%¥cluster¥nodes¥node1
        %ESP_HOME%¥bin¥esp_server --cluster-node node1.xml
      ```

   2. CCL をコンパイルして CCX を作成します。

      ```
      %ESP_HOME%¥bin¥esp_compiler -i model.ccl -o model.ccx
      ```

   3. プロジェクトをクラスタに展開します。

      ```
      %ESP_HOME%¥bin¥esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --add_project --workspace-
      name=w1 --project-name=p1 --ccx=model.ccx
      ```

   4. 展開したクラスタをサーバで起動します。

      ```
      %ESP_HOME%¥bin¥esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --start_project --
      workspace-name=w1 --project-name=p1
      ```

   UNIX：

   1. サンプル・クラスタを起動します。

      ```
      cd $ESP_HOME/cluster/nodes/node1
        $ESP_HOME/bin/esp_server --cluster-node node1.xml
      ```

   2. CCL をコンパイルして CCX を作成します。

      ```
      $ESP_HOME/bin/esp_compiler -i model.ccl -o model.ccx
      ```

   3. プロジェクトをクラスタに展開します。

      ```
      $ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --add_project --workspace-
      name=w1 --project-name=p1 --ccx=model.ccx
      ```

   4. 展開したクラスタをサーバで起動します。

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --start_project --
workspace-name=w1 --project-name=p1
```

2. アダプタを起動します。

オペレーティング・システム	手順

UNIX	端末ウィンドウを開き、次を入力。
	`cd $ESP_HOME/adapters/flex/bin`
	`./adapter.sh <configuration file path> start`

Windows	コマンド・ウィンドウを開き、次を入力。
	`cd %ESP_HOME%/adapters/flex/bin`
	`adapter.bat <configuration file path> start`

参照：

- *start コマンド* (190 ページ)

フレックス・アダプタのステータスの確認

フレックス・アダプタのステータスをコマンド・ラインから確認するには、**status** コマンドを実行します。

オペレーティング・システム	手順

UNIX	端末ウィンドウを開き、次を入力。
	`cd $ESP_HOME/adapters/flex/bin`
	`./adapter.sh <configuration file path> status`

Windows	コマンド・ウィンドウを開き、次を入力。
	`cd %ESP_HOME%/adapters/flex/bin`
	`adapter.bat <configuration file path> status`

アダプタのステータスが、実行中または停止中のいずれかとして表示されます。

参照：

- *status コマンド* (191 ページ)

フレックス・アダプタの停止
フレックス・アダプタをコマンド・ラインから停止するには、**stop** コマンドを実行します。

オペレーティング・システム	手順

UNIX	端末ウィンドウを開き、次を入力。
	`cd $ESP_HOME/adapters/flex/bin`
	`./adapter.sh <configuration file path> stop`

Windows	コマンド・ウィンドウを開き、次を入力。
	`cd %ESP_HOME%/adapters/flex/bin`
	`adapter.bat <configuration file path> stop`

参照：
- *stop コマンド* (191 ページ)

例： サブスクリプション要求の送信
サブスクリプション要求をアダプタに送信し、Event Stream Processor からのストリーム・レコードを Web ブラウザで表示します。

前提条件
- Web サーバ (デフォルト・ポートは 80)、Flash ポリシー・サーバ (デフォルトは 843)、Flash プラグインが組み込まれた Web ブラウザがインストールされている。
- Web サーバとポリシー・サーバは、アダプタがインストールされているのと同じマシン上で実行している。
- `example.swf` ファイルがアダプタの `example` ディレクトリから Web ブラウザのコンテンツ領域にコピーされている。
- アダプタがインストールされているマシンとは異なるマシンで Web サーバとポリシー・サーバが実行している場合、上記のポートが、Web ブラウザが実行しているマシンからの TCP 接続に対して開かれている。
- フレックス・サーバのポートのデフォルトは、23456 である。
- Web ブラウザを、同じマシン、または異なるマシン上で使用できる。

手順

1. `adapter.sh` スクリプトを編集します。
2. JAVA_HOME 環境変数を、Java Runtime Environment (JRE) がインストールされているディレクトリに設定します。
3. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`
Windows	コマンド・ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.bat` 2. プロジェクトをクラスタに追加し、クラスタ上で起動。 `start_project.bat`

4. アダプタを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./adapter.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `adapter.bat`

5. アダプタの初期化が完了するまで 5 ～ 10 秒待機します。
6. ストリーム・レコードをアップロードします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./upload.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `upload.bat`

7. Web ブラウザで、`example.swf` ファイルを参照します。例を示します。

```
http://localhost:80/sybase/example.swf
```

8. 次のストリーム・レコードがブラウザ・ウィンドウに表示されます。

```
Stream = Stream1
Opcode = i
Symbol = IBM
Price = 12.50
```

HTTP アウトプット・アダプタ

アダプタのタイプ： httpplugin。 Sybase Event Stream Processor の HTTP アダプタは、Event Stream Processor からのデータを外部クライアントにパブリッシュします。

HTTP アダプタは次の機能を提供します。

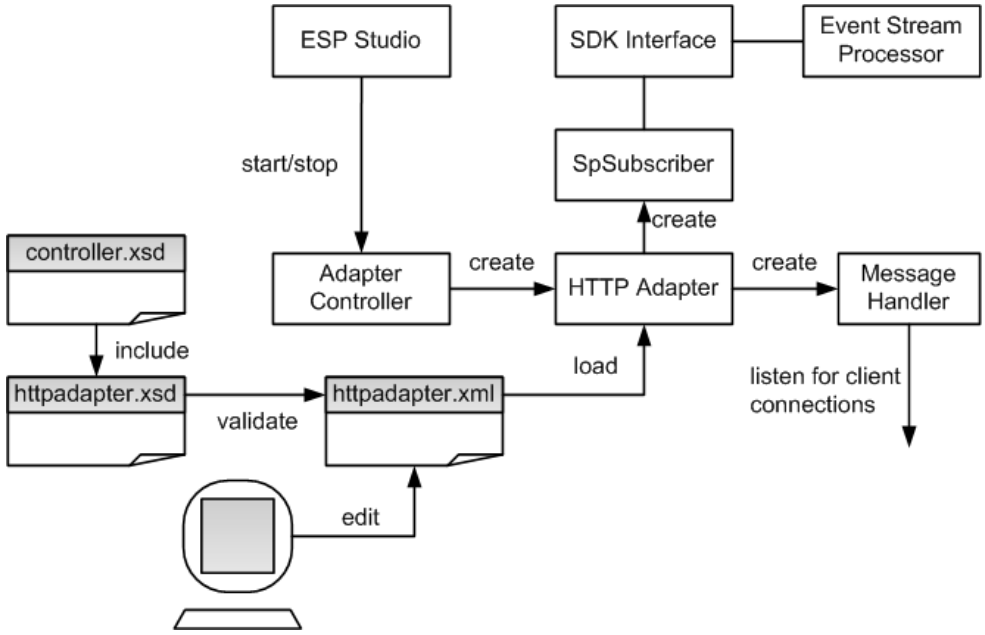
- 内部の HTTP サーバを実行し、クライアント接続を受信して受け付ける。
- クライアント要求から SQL クエリを抽出し、ストリームにサブスクライブする。
- ストリーム・レコードを XML に変換し、チャンク・コーディングされた HTTP 応答に XML を格納してクライアントに送信する。

制御フロー

アダプタは設定をファイル (たとえば、`adapter.xml`) からロードし、アダプタ・スキーマ (`httpadapter.xsd`) に対して検証します。このスキーマは、API 全体のコントローラ・スキーマ (`controller.xsd`) で構成されます。

スキーマは編集できません。

図 8 : HTTP アダプタ制御フロー



アダプタ・コントローラは、アダプタのインスタンスを作成し、ユーザ・コマンドを受け取り、実行します。 アダプタ・コントローラは、**start**、**stop**、**status** のコマンドを実行できます。

start コマンド

start コマンドは、アダプタの「コマンドと制御」のインタフェースを設定して起動します。このコマンドの実行によって、メッセージ・ハンドラがクライアント接続の受信を開始します。また、SpSubscriber コンポーネントが Event Stream Processor に SDK インタフェースを介して接続されます。

アダプタの実行中のインスタンスが存在するときに **start** コマンドを実行すると、このコマンドは無視され、警告が送信されます。

参照：

- *HTTP アダプタの起動* (215 ページ)

stop コマンド

stop コマンドは、SpSubscriber を Event Stream Processor から切断します。これによって、メッセージ・ハンドラは、既存のクライアントへの HTTP 応答を終了し、それらを切断します。また、新しいクライアントからの接続の受信を停止し、アダプタ・プロセスを終了します。

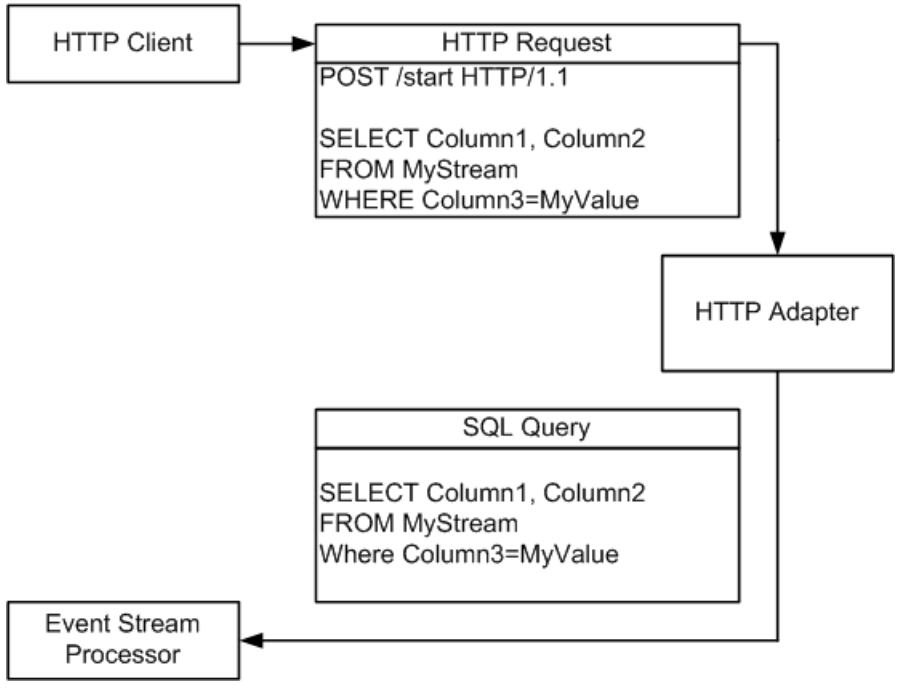
アダプタの実行中のインスタンスが存在しないときに **stop** コマンドを実行すると、コマンドは無視され、警告が送信されます。

参照：
- *HTTP アダプタの停止* (217 ページ)

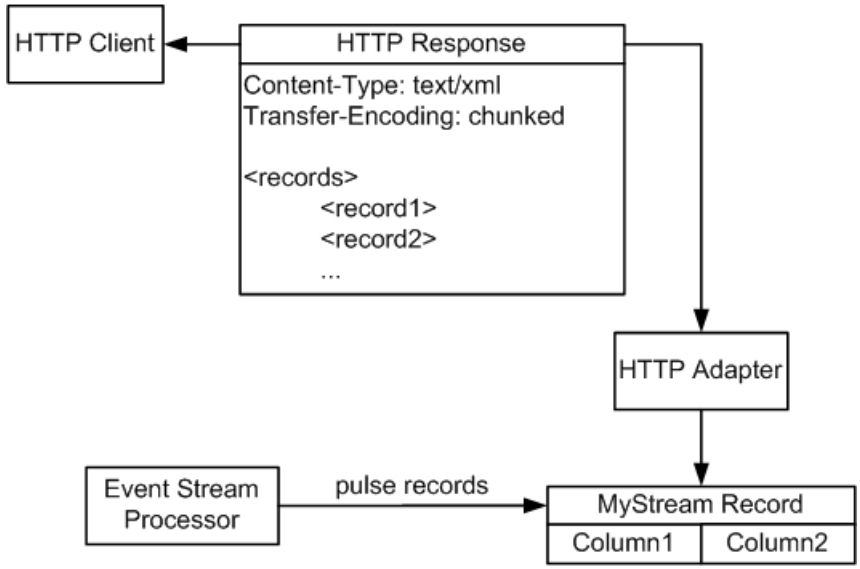
status コマンド
status コマンドは、アダプタのステータスを報告します。アダプタ・コントローラが次のステータスを表示します。 実行中または停止中のいずれか。

参照：
- *HTTP アダプタのステータスの確認* (216 ページ)

メッセージ・フロー
アダプタと HTTP クライアントとの間のメッセージ・フローは、クライアントが、**start** コマンドと Event Stream Processor への SQL で構成された本体が格納された POST 要求を送信すると開始されます。



対応するストリームでの変更は、XML でフォーマットされ、チャンク・コーディングされた HTTP 応答として、パルスを使用して HTTP クライアントに戻されます。



パルス期間は、アダプタ設定で指定されます。 フェールオーバが発生すると、SDK API はメッセージを失うことなく、設定に従って、パブリッシュ先を予備の Event Stream Processor インスタンスに切り替えます。

参照：
- *Event Stream Processor のパラメータ* (209 ページ)

JAVA_HOME 環境変数の設定
Java ディレクトリを指すように JAVA_HOME 環境変数を設定します。

前提条件
Java Runtime Environment のバージョン 1.6.0_26 以降をインストールする。

手順
JAVA_HOME 環境変数を、Java Runtime Environment 1.6.0_26 以降がインストールされているディレクトリ・パスに設定します。

設定
HTTP アダプタ用の設定情報を以下に示します。

HTTP アダプタのディレクトリ
アダプタのディレクトリには、設定ファイル、テンプレート、例、JAR ファイルなど、アダプタに関連するすべてのファイルがあります。

```
README.txt Quick Guide
ReleaseNotes.txt Release Notes
```

```
bin/
adapter.bat Standalone adapter startup script
adapter.sh Standalone adapter startup script
adapter-plugin.bat Plug-in connector startup script
adapter-plugin.sh Plug-in connector startup script

config/
controller.xsd Controller schema
log4j.properties Sample logging configuration
httpadapter.xsd Adapter schema
login.config Authentication configuration

example/ Working example

lib/
esp_http_adapter.jar http adapter library

javadoc/

adapterapi/ Adapter API Javadoc
httpadapter/ Http Adapter Javadoc

Common jars are located:

$ESP_HOME/adapters/jar

activation.jar Java mail library
adapterapi.jar Adapter API code
axis.jar Webservices jar
commons-codec-1.3.jar Required by SDK API
commons-discovery-0.2.jar
commons-logging-1.1.jar Logging library
esp_java_sdk-0.2.jar ESP SDK library
jaxrpc-api-1.1.jar Required by ESP SDK
log4j-1.2.14.jar Logging library
mail.jar Java mail library
saaj-api-1.3.jar Webservices jar
ws-commons-util-1.0.2.jar Required by ESP SDK
wsdl4j-1.5.1.jar
xercesImpl.jar XML parser library
xmlrpc-client-3.1.3.jar Required by ESP SDK
xmlrpc-common-3.1.3.jar Required by ESP SDK
xmlrpc-server-3.1.3.jar Required by ESP SDK
```

スキーマと設定ファイル

アダプタ設定はファイルからロードされ、アダプタのスキーマに対して検証されます。

`example`フォルダには、サンプルのアダプタ設定ファイルがあります。 有効な設定ファイルを提供し、アダプタ設定がアダプタ・スキーマに対して正しく検証されるようにする必要があります。

アダプタ・コントローラ・パラメータ

controllerPort パラメータは、アダプタの「コマンドと制御」のポートを指定します。

パラメータ名	タイプ	説明
controllerPort	`positive integer`	(必須) アダプタの「コマンドと制御」のポートを指定。 ユーザ・コマンドは、localhost 上のこのポートに送信されます。

Event Stream Processor のパラメータ

Event Stream Processor のパラメータは、Event Stream Processor と HTTP アダプタとの間の通信を設定します。

これらのパラメータは、`controller.xsd` ファイル (`config` ディレクトリ内) で定義されます。

パラメータ名	タイプ	説明
espAuthType	`string`	(必須) Event Stream Processor への認証に使用される方法を指定。 有効な値は、以下のとおりです。 • **server_rsa –** キーストアを使用する RSA 認証 • **user_password –** Kerberos 認証と LDAP 認証 • **none –** 認証なし アダプタがスタジオ・プラグインとして動作している場合、**espAuthType** は **Authentication Mode** スタジオ起動パラメータで上書きされます。
espUser	`string`	(必須) Event Stream Processor にログインするために必要なユーザ名を指定。 none (**espAuthType** を参照してください) 以外のすべての認証スキームで必要です。 デフォルト値はありません。

パラメータ名	タイプ	説明
espPassword	`string`	(必須) Event Stream Processor にログインするために必要なパスワードを指定。 none (**espAuthType** を参照してください) 以外のすべての認証スキームで必要です。 **espPassword** 値が暗号化されているかどうかを示す "encrypted" 属性も指定します。デフォルト値は false です。 true に設定すると、パスワード値は **espRSAKeyStore** と **espRSAKeyStorePassword** を使用して復号化されます。
espProjectUri	`string`	(必須) Event Stream Processor クラスタに接続するための完全なプロジェクト URI を指定。たとえば、`esp://localhost:19011/ws1/p1` のように記述します。
pulseInterval	`non-negative integer`	(必須) 期間を秒単位で指定。この期間では、アウトバウンド・レコードの変更は Event Stream Processor によって結合され、アダプタによって単一のイベントとして受信されるようになります。 設定されていないか、0 に設定されると、レコードの変更は、発生するごとに個々に受信されます。
espHeartbeatPeriod	`positive integer`	(オプション) 秒数を指定。この秒数が経過すると、アダプタは次のハートビートを Event Stream Processor に送信します。 Event Stream Processor が連続して 2 回のハートビート受信に失敗すると、アダプタがパブリッシュしたすべてのレコードが失効とマーク付けされます。 デフォルト値は 10 です。
recordQueueCapacity	`positive integer`	(オプション) レコード・キューの容量を指定。デフォルト値は、4096 です。
maxPubPoolSize	`positive integer`	(オプション) レコード・プールの最大サイズを指定。 レコード・プーリングは、パブリッシュ処理の高速化に役立ちます。デフォルト値は 256 です。

パラメータ名	タイプ	説明
maxPubPoolTime	`positive integer`	(オプション) レコードがパブリッシュされるまでにプールに存在可能な最大時間をミリ秒単位で指定。 設定されていない場合、プーリング時間は無制限で、プーリング方式は **maxPubPoolSize** によって制御されます。 デフォルト値はありません。
useTransactions	`boolean`	(オプション) true に設定すると、プールされたメッセージがトランザクションで Event Stream Processor にパブリッシュされる。 false に設定すると、エンベロープでパブリッシュされます。デフォルト値は false です。
espRSAKeyStore	`string`	(場合に応じて必須) RSA キーストアのロケーションを指定し、パスワード値を復号化するために使用される。 **espAuthType** が server_rsa に設定されている、**espPassword** の暗号化属性が true に設定されている、または両方の場合に必要です。
espRSAKeyStorePassword	`string`	(場合に応じて必須) キーストア・パスワードを指定し、パスワード値を復号化するのに使用される。**espAuthType** が server_rsa に設定されている、**espPassword** の暗号化属性が true に設定されている、または両方の場合に必要です。
espEncryptionAlgorithm	`string`	(オプション) **espPassword** の暗号化属性が true に設定されると使用される。 ブランクのままにすると、RSA がデフォルトとして使用されます。

参照：
- *メッセージ・フロー* (206 ページ)

HTTP サーバ設定
httpPort パラメータと **contentType** パラメータは、HTTP サーバ設定を指定します。

パラメータ	タイプ	説明
httpPort	`integer`	(必須) アダプタが HTTP サーバを実行するポートを指定。

パラメータ	タイプ	説明
contentType	`string`	(必須) HTTP 応答のコンテンツ・タイプを指定。 アダプタは、テキスト／プレーンとテキスト／html のコンテンツ・タイプをサポートします。

HTTP 設定ファイルのサンプル

HTTP アダプタ用のサンプル設定ファイル (`adapter.xml`) を以下に示します。

このファイルは、`example` にあります。

```
<adapter>
- <!--  Adapter Controller
  -->
- <controller>
  <controllerPort>13579</controllerPort>
  </controller>
- <!--  Streaming platform settings
  -->
- <esp>
- <espConnection>
  <espHost>localhost</espHost>
  <espPort>22000</espPort>
- <!--     <espProjectUri>esp://localhost:19011/ws1/p1</
espProjectUri>
  -->
  </espConnection>
- <espSecurity>
  <espUser>espuser</espUser>
  <espPassword encrypted="false">espuser</espPassword>
  <espAuthType>none</espAuthType>
- <!--
    <espRSAKeyFile>/keyfilepath/espuser.private.der</espRSAKeyFile>
    <espRSAKeyStore>/keystore/keystore.jks</espRSAKeyStore>
    <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword>

  -->
  <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
  </espSecurity>
  <maxPubPoolSize>1</maxPubPoolSize>
  </esp>
- <!--  HTTP specific
  -->
- <http>
  <httpPort>23456</httpPort>
  <contentType>text/html</contentType>
  </http>
  </adapter>
```

HTTP アウトプット・アダプタ

HTTP アウトプット・アダプタは、Web ブラウザなどのクライアント・アプリケーションから、HTTP 要求でラッピングされた SQL クエリを受信し、チャンクコーディングされたストリーム・コンテンツをクライアントに返信します。

すべてのソース・ストリームでアウトバウンド・データ・ロケーションとしてアダプタを設定できます。 認証方法が、Event Stream Processing 標準 (none、rsa、または gssapi) に設定されます。 このアダプタを使用するには、Sybase HTTP アダプタのバージョン 1.0 以降がインストールされている必要があります。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Connector Directory Path	**baseDir**	`directory`	(必須) アダプタ・インストール・ディレクトリへの絶対パスを指定する。 このプロパティは、**Connector Remote Directory Path** プロパティが指定されていると無視されます。 デフォルト値はありません。
Configuration File Path	**configFilePath**	`configFilename`	(必須) アダプタ設定ファイルへの絶対パスを指定する。 このプロパティは、**Remote Configuration Path** プロパティが指定されていると無視されます。 デフォルト値はありません。
Connector Remote Directory Path	**remoteBaseDir**	`string`	(詳細) アダプタ・リモート・ベース・ディレクトリのパスを指定 (リモート実行のみ)。 このプロパティが指定されている場合、**Connector Directory Path** プロパティは無視されます。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Remote Configuration File Path	**remoteConfigFilePath**	`string`	(詳細) アダプタ・リモート設定ファイルのパスを指定 (リモート実行のみ)。 このプロパティが指定されている場合、**Configuration File Path** プロパティは無視されます。 デフォルト値はありません。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

ロギング

アダプタは Apache `log4j` API を使用して、エラー、警告、情報、デバッグ・メッセージをログ記録します。

`log4j.properties` ファイルには、ログ記録設定があります。 サンプル `log4j.properties` ファイルが、アダプタ配布と共に提供されます。

該当するロギング・レベルを `log4j.properties` ファイルで設定します。 ログ記録レベルを DEBUG に設定すると、ログ・ファイルが非常に大きくなることがあります。 デフォルトのレベルは、INFO です。 ロー IDC メッセージが、DEBUG レベルでログ記録されます。 ログ記録設定の詳細については、http://logging.apache.org/log4j を参照してください。

オペレーション

HTTP Output アダプタをコマンド・ラインから操作できます。

HTTP アダプタの起動

HTTP アダプタをコマンド・ラインから起動するには、Event Stream Processor を起動して、**start** コマンドを実行します。

1. Event Stream Processor を起動します。

   Windows：

   1. サンプル・クラスタを起動します。

      ```
      cd %ESP_HOME%¥cluster¥nodes¥node1
        %ESP_HOME%¥bin¥esp_server --cluster-node node1.xml
      ```

   2. CCL をコンパイルして CCX を作成します。

      ```
      %ESP_HOME%¥bin¥esp_compiler -i model.ccl -o model.ccx
      ```

   3. プロジェクトをクラスタに展開します。

      ```
      %ESP_HOME%¥bin¥esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --add_project --workspace-
      name=w1 --project-name=p1 --ccx=model.ccx
      ```

   4. 展開したクラスタをサーバで起動します。

      ```
      %ESP_HOME%¥bin¥esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --start_project --
      workspace-name=w1 --project-name=p1
      ```

   UNIX：

   1. サンプル・クラスタを起動します。

      ```
      cd $ESP_HOME/cluster/nodes/node1
        $ESP_HOME/bin/esp_server --cluster-node node1.xml
      ```

   2. CCL をコンパイルして CCX を作成します。

      ```
      $ESP_HOME/bin/esp_compiler -i model.ccl -o model.ccx
      ```

   3. プロジェクトをクラスタに展開します。

      ```
      $ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --add_project --workspace-
      name=w1 --project-name=p1 --ccx=model.ccx
      ```

   4. 展開したクラスタをサーバで起動します。

      ```
      $ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --start_project --
      workspace-name=w1 --project-name=p1
      ```

2. アダプタを起動します。

オペレーティング・システム	手順

UNIX	端末ウィンドウを開き、次を入力。
	`cd $ESP_HOME/adapters/http/bin`
	`./adapter.sh <configuration file path> start`

Windows	コマンド・ウィンドウを開き、次を入力。
	`cd %ESP_HOME%/adapters/http/bin`
	`adapter.bat <configuration file path> start`

参照：

- *start コマンド* (205 ページ)

HTTP アダプタのステータスの確認

HTTP アダプタのステータスをコマンド・ラインから確認するには、**status** コマンドを実行します。

オペレーティング・システム	手順

UNIX	端末ウィンドウを開き、次を入力。
	`cd $ESP_HOME/adapters/http/bin`
	`./adapter.sh <configuration file path> status`

Windows	コマンド・ウィンドウを開き、次を入力。
	`cd %ESP_HOME%/adapters/http/bin`
	`adapter.bat <configuration file path> status`

アダプタのステータスが、実行中または停止中のいずれかとして表示されます。

参照：

- *status コマンド* (206 ページ)

HTTP アダプタの停止
HTTP アダプタをコマンド・ラインから停止するには、**stop** コマンドを実行します。

オペレーティング・システム	手順

UNIX	端末ウィンドウを開き、次を入力。
	`cd $ESP_HOME/adapters/http/bin`
	`./adapter.sh <configuration file path> stop`

Windows	コマンド・ウィンドウを開き、次を入力。
	`cd %ESP_HOME%/adapters/http/bin`
	`adapter.bat <configuration file path> stop`

参照：
- *stop コマンド* (205 ページ)

例： データの送信、受信、表示
アダプタ配布と共に提供される実際に動作する例を使用して、SQL クエリをアダプタに送信する方法、XML でフォーマットされたストリーム・データを Event Stream Processor から受信する方法、ストリーム・データを Web ブラウザで表示する方法について習得します。

前提条件
ネットワーク接続が必要です。

手順

1. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`

オペレーティング・システム	手順

Windows	コマンド・ウィンドウを開く。
	1. サンプル・クラスタを起動。
	`start_server_cluster.bat`
	2. プロジェクトをクラスタに追加し、クラスタ上で起動。
	`start_project.bat`

2. `adapter.sh` スクリプトを編集します。
3. JAVA_HOME 環境変数を、Java Runtime Environment (JRE) がインストールされているディレクトリに設定します。
4. アダプタを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./adapter.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `adapter.bat`

5. アダプタの初期化が完了するまで 5 ～ 10 秒待機します。
6. ストリーム・レコードのアップロードを開始します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./upload.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `upload.bat`

7. HTTPAdapterClient.html ページを Web ブラウザにロードします。
8. たとえば次のような、有効な SQL クエリを入力します。

```
SELECT * FROM Stream1
```

注意： 少量のレコードを返すクエリが、一部のブラウザでは表示されないことがあります。これは、それらのブラウザでは、バッファに一定量のデータが蓄積されてからでないと、データを表示しないためです。 たとえば、次の SQL クエリの場合：

```
SELECT * FROM Stream1
where intcol = 10
```

- Mozilla FireFox ブラウザ： レコードが表示される。
- Google Chrome ブラウザ： レコードは表示されない。
- Internet Explorer ブラウザ： レコードは表示されない。

```
SELECT * FROM Stream1
where intCol > 10
```

(デバッグ・モードでの実行)

– Mozilla FireFox ブラウザ： レコードはブラウザに送信されてから、表示される。

– Google Chrome ブラウザ： レコードは、5 つ目のレコードがブラウザに送信されてから、表示される。

– Internet Explorer ブラウザ： レコードは、20 番目のレコードがブラウザに送信されてから、表示される。

9. [Submit] をクリックします。
10. Web ブラウザ・ウィンドウに送信されるレコードを書き留めます。

KDB アダプタ

Adapter types: KDBInput、KDBOutput。 Sybase Event Stream Processor の KDB インプット・アダプタと KDB アウトプット・アダプタを使用すると、データを kdb+ データベースから Event Stream Processor プロジェクトにロードし、Event Stream Processor プロジェクトからの出力を kdb+ データベースに格納できます。

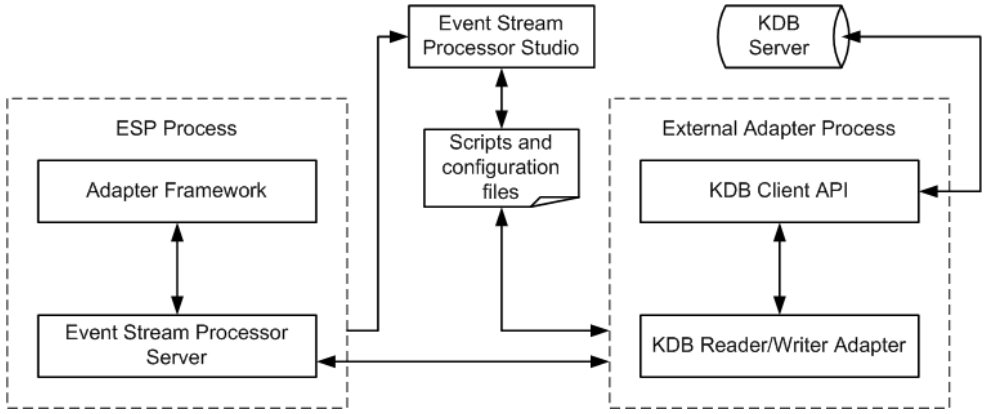
注意： KDB インプット・アダプタと KDB アウトプット・アダプタは、Solaris AMD プラットフォームをサポートしません。

制御フロー

KDB インプット・アダプタと KDB アウトプット・アダプタの運用には、一連のスクリプト・ファイルを使用します。

アダプタ・スクリプトでは、次の基本コマンドが使用されます。

図 9 : KDB アダプタの制御フロー



start コマンド

start コマンドは、KDB インプット・アダプタ、KDB アウトプット・アダプタ、KDB クライアント API、KDB データベース・サーバを起動し、これらを SDK インタフェースを介して Event Stream Processor に接続します。

stop コマンド

stop コマンドは、KDB インプット・アダプタと KDB アウトプット・アダプタを停止し、KDB データベース・サーバと Event Stream Processor との間の接続を閉じます。

KDB アダプタ用のデータ型マッピング

Event Stream Processor のデータ型は KDB のデータ型にマップされ、KDB のデータ型は Event Stream Processor のデータ型にマップされます。

KDB データ型から ESP データ型へのマッピング

KDB データ型は ESP データ型にマップされます。

KDB データ型	文字	ESP データ型
boolean	b	`boolean`
byte	x	`integer`
short	h	`integer`
int	i	`integer`
long	j	`long`

KDB データ型	文字	ESP データ型
real	e	`float`
float	f	`float`
symbol	s	`string`
date	d	`date`
datetime	z	`timestamp`
time	t	`timestamp`
月	m	`date`
分	u	`interval`
秒	v	`interval`

ESP データ型から KDB データ型へのマッピング
Event Stream Processor データ型は、KDB データ型にマップされます。

ESP データ型	文字	KDB データ型
`integer`	i	int
`long`	j	long
`float`	f	float
`money`	f	float
`string`	s	symbol
`date`	z	datatime
`timestamp`	z	datatime
`bigdatetime`	j	long
`interval`	j	long
`boolean`	b	boolean

KDB インプット・アダプタ

KDB インプット・アダプタは、kdb または kdb+tick のデータベース・テーブルを読み込みます。

デフォルトでは、アダプタは名前でフィールドを一致させ (大文字小文字は区別されません)、ソースの kdb+tick テーブルとターゲット・ストリームとの間のマッピングを決定します。 KDB アウトプット・アダプタは、カスタム・フィールドマッピングもサポートします。

このアダプタは、スキーマ検出をサポートします。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは KDBInput です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
KDB Server	**server**	`string`	(必須) データベース・サーバ・マシンの名前または IP アドレスを指定。 デフォルト値は、localhost です。
KDB Port	**port**	`range`	(必須) データベース・リスナの IP ポートを指定。 デフォルト値は 5001 です。最小値は 0、最大値は 65535 です。
Event Stream Processor User ID	**espUser**	`string`	(オプション) Event Stream Processor に接続するためのユーザ名を指定。 デフォルト値は t です。
Event Stream Processor Password	**espPassword**	`password`	(オプション) Event Stream Processor のユーザ ID のパスワードを指定。 **注意：** RSA 認証が使用されている場合、このプロパティはブランクのままにできます。 デフォルト値は t です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Authentication	**authentication**	`choice`	(オプション) kdb または kdb+tick のデータベース・テーブルへの認証で使用される方法を指定。 有効な値は次のとおりです。 • `value="none" label="None"` • `value="pam" label="PAM"` • `value="rsa" label="RSA"` • `value="gssapi" label="Kerberos V5"` デフォルト値はありません。
Project URI	**projectUri**	`string`	(オプション) クラスタ環境でプロジェクトに接続するための URI を指定。 デフォルト値はありません。
RSA Key File	**rsaKeyFile**	`filename`	(オプション) RSA プライベート・キーのファイル名とロケーションを指定。 デフォルト値はありません。
KDB User	**user**	`string`	(オプション) データベース接続用のユーザ ID を指定。 デフォルト値はありません。
KDB Password	**password**	`password`	(オプション) データベース接続用のパスワードを指定。 デフォルト値はありません。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Source Table	**table**	`tables`	(詳細) データの取得元のソース・テーブルの名前を指定。 **注意：** このプロパティは、SQL クエリ文をサポートします。 デフォルト値はありません。
Field Mapping	**permutation**	`string`	(詳細) Event Stream Processor の内部フィールドと外部フィールドとの間のマッピングを指定。 フォーマットは次のとおりです。 `ESPColumn1=KDBColumn1:ESPColumn2=KDBColumn2...`. デフォルト値はありません。
Streaming Mode	**mode**	`choice`	(詳細) アダプタは kdb+tick データベースに接続してストリーミング・データを読み込むのか、または提供されているクエリを実行して結果を Event Stream Processor にフィードするのかを指定。 有効な値は次のとおりです。 • `value="stream", label="Stream"` • `value="pull", label="One time pull"` デフォルト値は、"stream" です。
Polling Interval	**pollInterval**	`int`	(詳細) クエリをプル・モードで再実行するまでに待機する時間を秒単位で指定。 0 に設定すると、クエリは再実行されません。デフォルト値は 0 です。
Block Size	**blockSize**	`int`	(詳細) 擬似トランザクションになる、ブロックあたりのレコード数を指定。 デフォルト値は 64 です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Async Mode	**async**	`boolean`	(詳細) true に設定すると、アダプタは、データを受信する Event Stream Processor からの確認応答を待たない。デフォルト値は false です。
Encrypt Connection	**encrypt**	`boolean`	(詳細) true に設定すると、Event Stream Processor とアダプタとの間のトラフィックが暗号化される。デフォルト値は false です。
Print Debug Info	**デバッグ**	`boolean`	(詳細) アダプタが追加のデバッグ情報をコンソールに表示するかどうかを指定。デフォルト値は false です。
Use Transaction Blocks	**useTransaction**	`boolean`	(詳細) パフォーマンスを向上させるために、データを Event Stream Processor に送信するときにエンベロープの代わりにトランザクション・ブロックを使用。デフォルト値は false です。
Connection Retries	**connectionRetries**	`int`	(詳細) 接続が失われたときに接続を試行する回数を指定。デフォルト値は 1 です。
Event Stream Processor Host Name	**gatewayHost**	`string`	(詳細) 明示的なゲートウェイ・ホスト名を指定。 デフォルト値はありません。
パラメータ・ファイル	**x_paramFile**	`filename`	(詳細) パラメータを外部プロセスに渡すための、パラメータを書き込むファイルを指定。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Parameter File Format	**x_paramFormat**	`choice`	(詳細) 外部プロセスが期待するパラメータのフォーマットを指定。有効な値は以下のとおりです。 • `value = prop, label = "Java properties"` • `value= shell, label="Unix shell assignments"` • `value = xml, label = "Simple XML"` デフォルト値は、"prop" です。

既知の制限事項：

- アダプタが接続を試行するときに kdb+tick データベースが実行していない場合、kdb+tick データベースが起動するまで、アダプタは無期限に待機する。

  注意： この問題が発生するのは、kdb+tick データベースと Event Stream Processor が異なるマシン上で実行している場合のみです。
- データベースへの接続が切断されると、接続の切断と再確立との間に発生したすべての更新は失われる。

KDB アウトプット・アダプタ

KDB アウトプット・アダプタは、Event Stream Processor からのストリーム・データを KDB+tick データベース・テーブルにパブリッシュします。

デフォルトでは、アダプタは名前でフィールドを一致させ (大文字小文字は区別されません)、ソースの KDB+tick テーブルとターゲット・ストリームとの間のマッピングを決定します。 KDB アウトプット・アダプタは、カスタム・フィールド・マッピングをサポートします。

CCL **ATTACH ADAPTER** 文を使用してアダプタをアタッチする場合、アダプタ・タイプを指定する必要があります。 このアダプタのタイプは KDBOutput です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
KDB Server	**server**	`string`	(必須) データベース・サーバ・マシンの名前または IP アドレス。 デフォルト値はありません。
KDB Port	**port**	`range`	(必須) データベース・リスナの IP ポート。 デフォルト値は 5001 です。最小値は 0、最大値は 65535 です。
Event Stream Processor User ID	**espUser**	`string`	(オプション) Event Stream Processor に接続するためのユーザ名。 デフォルト値はありません。
Event Stream Processor Password	**espPassword**	`password`	(オプション) Event Stream Processor のユーザ ID のためのパスワード。RSA 認証を使用している場合、空に設定できます。 デフォルト値はありません。
Authentication	**authentication**	`choice`	(オプション) 使用する認証方法。 有効な値は次のとおりです。 • `value="none" label="None"` • `Value value="rsa" label="RSA"` • `Value value="gssapi" label="Kerberos V5"` デフォルト値はありません。
Project URI	**projectUri**	`string`	(オプション) クラスタ環境でプロジェクトに接続するための URI を指定。 デフォルト値はありません。
RSA Key File	**rsaKeyFile**	`filename`	(オプション) RSA プライベート・キーのファイル名とロケーション。 デフォルト値はありません。
KDB User	**user**	`string`	(オプション) データベース接続用のユーザ ID。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
KDB Password	**password**	`password`	(オプション) データベース接続用のパスワード。 デフォルト値はありません。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。
Target Table	**table**	`tables`	(詳細) 更新するターゲット・テーブルの名前。 デフォルト値はありません。
SQL Query	**query**	`string`	(詳細) 受信データをフィルタするための SQL クエリ。 デフォルト値はありません。
Field Mapping	**permutation**	`string`	(詳細) Event Stream Processor の内部フィールドと外部フィールドとの間のマッピング。 フォーマットは次のとおりです。`ESPColumn1=KDBColumn1:ESPColumn2=KDBColumn2...`. デフォルト値はありません。
Streaming Mode	**mode**	`choice`	(詳細) ストリーミング・モード。 有効な値は次のとおりです。 • `value="stream", label="user.u.upd"` • `value="push", label=”use update"` デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Async Mode	**async**	`boolean`	(詳細) true に設定すると、アダプタは、データを受信する Event Stream Processor からの確認応答を待たない。デフォルト値は false です。
Connection Retries	**connectionRetries**	`int`	(詳細) 接続が失われたときに接続を試行する回数。デフォルト値は 1 です。
KDB Batch size	**blockSize**	`int`	(詳細) 1 回の KDB 書き込みバッチの最大レコード数。デフォルト値は 5000 です。
BASE Data Only	**baseOnly**	`boolean`	(詳細) true に設定すると、接続時に存在するデータが取得されない。デフォルト値は false です。
Lossy Subscriber	**lossy**	`boolean`	(詳細) true に設定すると、接続がスローダウンした場合に Event Stream Processor によってレコードが削除される。デフォルト値は false です。
Pulse Interval	**pulsed**	`int`	(詳細) ゼロ以外の値に設定すると、サブスクリプションが、指定した間隔のパルス・モードで作成される。デフォルト値は 0 です。
Shine Through	**shine**	`boolean`	(詳細) true に設定すると、サブスクライバはシャイン・スルーを使用してデータを送信する。デフォルト値は false です。
Droppable Subscription	**droppable**	`boolean`	(詳細) true に設定すると、データがバックアップされた場合にサブスクリプションが Event Stream Processor によって削除される。デフォルト値は false です。
Preserve Transaction Blocks	**originalBlocks**	`boolean`	(詳細) true に設定すると、サブスクライブされたデータのトランザクション境界が Event Stream Processor によって維持される。デフォルト値は false です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Debug	**debug**	`boolean`	(詳細) true に設定すると、KDB データベース・サーバと Event Stream Processor との間の接続のデバッグ・メッセージが、他の Event Stream Processor ログと一緒に表示される。デフォルト値は false です。
Omitted Fields	**omitFields**	`string`	(詳細) 省略する KDB フィールドのカンマ区切りのリスト。 **注意：** 省略されたフィールドは送信されません。 デフォルト値はありません。
Ignored Fields	**ignoreFields**	`string`	(詳細) 無視する KDB フィールドのカンマ区切りのリスト。これらについては、null が送信されます。 デフォルト値はありません。
Quiet Mode	**quietMode**	`boolean`	(詳細) true に設定すると、KDB ログ・メッセージは標準のエラー出力に表示されない。デフォルト値は false です。
Encrypt Connection	**encrypt**	`boolean`	(詳細) true に設定すると、Event Stream Processor と KDB アダプタとの間のトラフィックが暗号化される。デフォルト値は false です。
Parameter File	**x_paramFile**	`filename`	(詳細) パラメータを外部プロセスに渡すための、パラメータを書き込むファイル。 デフォルト値は、`<temp>¥PARAMETER_FILE.txt` です。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Parameter File Format	**x_paramFormat**	`choice`	(詳細) 外部プロセスが期待するパラメータのフォーマット。 有効な値は以下のとおりです。 • `value = prop, label = "Java properties"` • `value = shell, label= "Unix shell assignments"` • `value = xml, label = "Simple XML"` デフォルト値は、"prop" です。

ログファイル・インプット・アダプタ

Sybase Event Stream Processor のログファイル・インプット・アダプタは、ログ・ファイルから読み込み、データをストリームに送信します。

ファイルから読み込まれたログ・レコードごとに、アダプタは、1 つのロー (メッセージ) をストリームに送信します。 ログ・ファイルは次のどのフォーマットでもかまいません。

- 共通ログ・フォーマット
- 複合ログ・フォーマット
- 拡張共通ログ・フォーマット

フォーマットを選択することによって、これらのフォーマットの 1 つを使用してログ・ファイルに書き込む Apache、Tomcat、IIS、他のデータソースからのログ・ファイルを読み込めます。 該当する .properties ファイルをカスタマイズして、ログ・ファイルを他のフォーマットで読み込むこともできます。

このアダプタは Java で記述されており、JavaBean を使用します。 このアダプタを使用するには、JavaBean と .properties ファイルに精通している必要があります。

このアダプタを使用して、ライブ・ログ・ファイルまたは履歴ログ・ファイルのいずれかから読み込めます。 履歴ログ・ファイルは完結しており、最初から最後まで 1 回で読み込めます。 ライブ・ログ・ファイルは、未完結で、データが連続して追加されます。

ライブ・ログ・ファイルには、ローテーティングとアドバンシングの 2 つのタイプがあります。 ローテーティング・ログ・ファイルでは、ログ・ファイルがいっぱいになると、ファイルに書き込みを行うプログラムが既存のファイルの名前を

変更し、元の名前で新しいファイルを作成します。 通常、新しいファイル名は、元のファイル名に基づき、サフィックスが追加されます。たとえば、元のファイル名が "access.log" の場合、名前が変更されたファイルは、"access.log.1"、"access.log.2" などとなります。 ログファイル・インプット・アダプタは、常に、元の名前のファイルを読み込みます。 古いファイルの名前が変更され、新しいファイルが作成されると、アダプタは先頭に戻って読み込みを実行します。

アドバンシング・ログ・ファイルの場合、ファイルがいっぱいになると、ログ・ファイル・ライタが新しいファイルを作成し、その新しいファイルへの書き込みを開始します。 命名規則は、通常、基本名とサフィックスの組み合わせです。サフィックスには、日付／時間または連番が使用されます。たとえば、"access-log.2007-01-01"、"access-log.2007-01-02" など、または "access.log.1"、"access.log.2" などです。 命名規則に関係なく、アダプタはこれらのログ・ファイルを最終変更日時の時間順で開きます。

設定

アダプタの `.properties` ファイルで値を設定することによって、ログファイル・インプット・アダプタを設定します。

たとえば、読み込むログ・ファイルの名前を指定するには、**Input.Filename** プロパティを設定します。 `.properties` ファイルの例が、製品に同梱されています。

このアダプタを使用すると、ライブ・ログ・ファイルまたは履歴ログ・ファイルを読み込めます。

- 履歴ログ・ファイルを読み込むには、ファイル名を **Input.Filename** プロパティで指定し、**Input.WaitForGrowth** プロパティを false に設定する。
- ライブ・ログ・ファイルを読み込むには、ローテーティングまたはアドバンシングに関係なく、**Input.WaitForGrowth** プロパティを true に設定する。 ログファイル・インプット・アダプタはファイルの最後まで読み込み、新しいデータがファイルに追加されると、そのデータを読み込みます。 ファイル・サイズがゼロまで小さくなる (古いログ・ファイルの名前が変更され、新しい空のログ・ファイルが作成される) と、ログファイル・インプット・アダプタは新しいファイルの先頭から引き続き読み込みを実行します。

プロパティ

.properties ファイルの重要なプロパティの一部をリストし、簡単に説明します。アダプタの設定可能なプロパティの完全で最新のリストについては、`example.properties` ファイルを参照してください。

`example .properties` ファイルには、共通ログ・フォーマットと複合ログ・フォーマットのカラム・リストのコメント化された例があります (このファイルの "Parse.Class" セクションを参照してください)。 拡張共通ログ・フォーマットを使用するには、カラム名とデータ型を Parse.Format.Common.Columns プロパティに追

加します。 ログ・ファイル・リーダは拡張と変更が可能であるので、独自のフォーマットを定義し、既存のフォーマットを変更して、設定に一致できます。 たとえば、Apache サーバが共通ログ・フォーマットまたは複合ログ・フォーマットで書き込むためのフォーマットを変更した場合、一致するようにプロパティ・ファイルを変更できます。 既存のエントリを変更することも、新しいエントリを作成することもできます。 たとえば、"MyUncommonFormat" という名前の新しいフォーマットを作成し、そのフォーマットのカラムを定義できます。

プロパティ名	タイプ	説明
Input.Filename	`string`	読み込み先のログ・ファイルの名前を表示。
Input.WaitForGrowth	`string`(true、false、ミリ秒数、またはミリ秒数とそれに続くカンマとナノ秒数)	false に設定すると、アダプタはファイルの末尾に到達するまで読み込みを実行し、到達すると終了する。 true に設定すると、アダプタはログ・ファイルからの読み込みを、100 ミリ秒の間隔で無限に繰り返します。100 ミリ秒はデフォルトの間隔です。 ミリ秒または「(ミリ秒),(ナノ秒)」のフォーマットを指定すると、サーバは指定された間隔でログ・ファイルからの読み込みを無限に繰り返します。たとえば、10 を指定すると、アダプタは 10 ミリ秒ごとに読み込みを実行し、0,500000 を指定するとアダプタは 500000 ナノ秒ごとに、または 1 秒間に 2000 回チェックを実行します。 読み込み間の実際の間隔は、指定した間隔の近似値となり、実際の間隔は、システムのハードウェア、オペレーティング・システム、負荷によって異なります。
Parse.Class	`string`	ログ・ファイルのカラムを解析するために使用される JavaBean を指定。
Parse.FormatName	`string`	ログ・ファイルのデータのフォーマットの名前を指定。 これには、共通または複合など事前に定義されたフォーマットの名前の 1 つを指定できますし、カスタム・フォーマットの名前も指定できます。

プロパティ名	タイプ	説明
Parse.{FormatName}.Columns (ここで {FormatName} は、Parse.FormatName によって提供される名前に置き換えられる)	`string`	ログ・ファイルの名前と、カラム名のデータ型で構成される。 このプロパティをカスタマイズする必要があります。
Parse.{FormatName}.DateColumn (ここで {FormatName} は、Parse.FormatName によって提供される名前に置き換えられる)	`string`	ログ・ファイルの日付カラムの名前で構成される。 この値は、日付カラムの実際の名前に一致する必要があります。
Parse.{FormatName}.TimeColumn (ここで {FormatName} は、Parse.FormatName によって提供される名前に置き換えられる)	`string`	(オプション) ログ・ファイルの時刻カラムの名前で構成される。 このカラムは、日付カラムで時刻が構成されていない場合のみ必須です。
Output.Uri	`string`	ローの送信先のストリームの URI。
Output.Columns	`string`	文字列は、スペースで区切られたカラム名のリストで構成される必要あり。これによって、出力を書き込み先のストリームのスキーマに一致できます。
Admin.Input	`string`	Admin.Input を stdin に設定すると、ログファイル・アダプタは stdin を読み込んで管理コマンドを探します。 現在サポートされている管理コマンドは、exit のみです。このコマンドは、入力ラインの exit、quit、x、または q の いずれによっても要求できます。このプロパティの主な目的は、デバッグを支援することです。 デフォルトは、管理要求のない Bean 用です。
Output.AuthType	`string`	(オプション) 認証タイプを指定。有効な値は次のとおりです。 None、UserPassword、または ServerRSA。 デフォルト値は、None です。

プロパティ名	タイプ	説明
Output.username	`string`	(場合に応じて必須) None 以外のすべての認証方法で必須。 UserPassword の場合は、サーバのユーザ名を指定します。 ServerRSA 認証方法については、キーストア・エイリアスを指定します。
Output.password	`string`	(場合に応じて必須) UserPassword 認証方法の場合に必須。 サーバのパスワードを指定します。
Output.passwordEncrypted	`string`	(場合に応じて必須) Output.password プロパティのパスワードが暗号化されるかどうかを指定。 有効な値は、true または false です。デフォルト値は false です。
Output.RSAKeyStore	`string`	(場合に応じて必須) キーストア・ファイルのロケーションを指定。 ServerRSA 認証方法または UserPassword (パスワードが暗号化される場合) で必要です。
Output.RSAKeyStoreAlias	`string`	(場合に応じて必須) キーストア・エイリアスを指定。 UserPassword 認証方法でパスワードが暗号化される場合に必要です。
Output.RSAKeyStorePassword	`string`	(場合に応じて必須) RSA キーストアのパスワードを指定。 ServerRSA 認証方法または UserPassword (パスワードが暗号化される場合) で必要です。
Parse.BigDatetime.Format	`string`	bigdatetime カラムのフォーマットを指定。 java.text.SimpleDateFormat を使用します。これは、マイクロ秒をサポートしません。 マイクロ秒を表示するには、文字列 UUUUUU が存在することを確認します。 **注意：** マイクロ秒の後に月があり、月の文字数が可変の場合、データは正しく解析されません。 たとえば、完全な月名が使用されると、September は July よりも文字数が多く、July には May よりも多くの文字数があります。

注意： 各入力ストリームには、アダプタによって設定されるローのタイムスタンプの代わりに現在のサーバのタイムスタンプ値を使用するかどうかを指定できるプロパティがあります (スタジオで、ストリームの [Properties] タブを参照してください)。 このストリーム・プロパティを true に設定すると、アダプタによって設定されたローのタイムスタンプが上書きされます。

コマンド・ラインからのアダプタの起動

ログファイル・インプット・アダプタをコマンド・ラインから起動するには、CLASSPATH 環境変数を設定し、**start** コマンドを実行します。

CLASSPATH 環境変数を設定するには、`createClasspath.sh` (UNIX、Linux の場合) または `createClasspath.bat` (Windows の場合) を使用します。

ほとんどの UNIX 系オペレーティング・システムで、スタートアップ・スクリプトは `/etc/init.d` にあります。 このようなシステムでアダプタを起動する場合、スクリプト `logfile_input.rc` を `/etc/init.d` ディレクトリにコピーして、必要な編集を行うことをおすすめします。

注意： `logfile_input.rc` スクリプトはサンプルで、Linux 上でのみ動作します。 これを別のプラットフォームで使用するには、このスクリプトをカスタマイズする必要があります。

`/etc/init.d` のないシステムでは、独自のスクリプトを開発してログファイル・インプット・アダプタを起動するための基本として `logfile_input.rc` を使用できます。

`logfile_input.rc` ファイルを介して実行できるコマンドは、以下のとおりです。

- **Start**
- **Stop**
- **Status**
- **Restart**

注意： `logfile_input.rc` スクリプトを実行する場合、`/etc/sysconfig`、`/var/run`、`/var/lock/sybsys` への書き込みパーミッションが必要です。

1. コマンド **source createClasspath.bat** (Windows の場合)、またはコマンド **source createClasspath.sh** (UNIX の場合) を実行して、CLASSPATH 環境変数を設定します。
2. アダプタをコマンド・ラインから実行するには、次のコマンドを実行します。

```
java -cp $CP -Dproperties=FILE.PROPERTIES
com.sybase.esp.adapters.logFileInput.Main
```

"D" と単語 "properties" の間にはスペースはありません。

NYSE Technologies インプット・アダプタ

アダプタのタイプ： wombatplugin。 NYSE Technologies MAMA 用の Sybase Event Stream Processor アダプタ (NYSE アダプタ) は、Wombat マーケット・データ・インフラストラクチャに接続するために使用されます。

NYSE アダプタは、Solaris SPARC プラットフォームをサポートしません。

NYSE アダプタは以下の機能を提供します。

- Wombat データ・フィードに接続し、セッションを開いて、サブスクリプションの作成とサインオフを実行する。
- Wombat メッセージを Event Stream Processor レコードに変換する。

NYSE アダプタを使用するには、個別のライセンスを購入する必要があります。このライセンスは、Sybase 製品ダウンロード・サイトから取得できます。 SySAM 猶予期間がサポートされていないので、有効なライセンスなしでは実行できません。

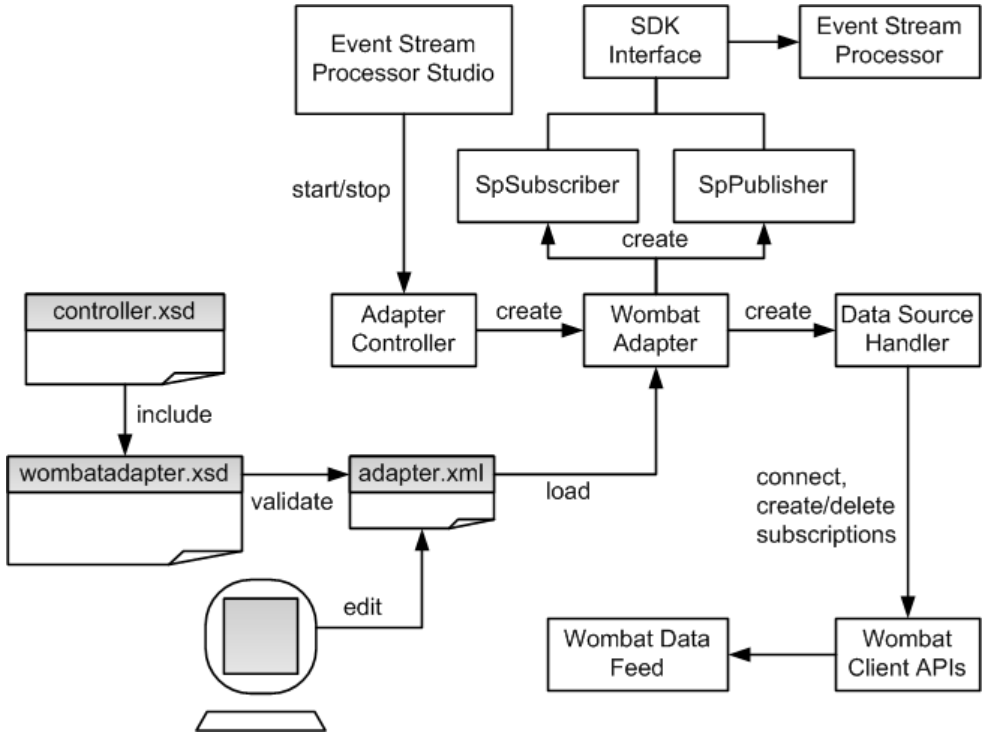
制御フロー

アダプタは設定をファイル (たとえば、`adapter.xml`) からロードし、アダプタ・スキーマ (`wombatadapter.xsd`) に対して検証します。このスキーマは、API 全体のコントローラ・スキーマ (`controller.xsd`) で構成されます。

スキーマは編集できません。

アダプタ・コントローラがアダプタのインスタンスを作成し、**start**、**stop**、**status** のコマンドを受け取り、実行します。

図 10 : NYSE Technologies アダプタの制御フロー



start コマンド

start コマンドは、アダプタの「コマンドと制御」のインタフェースを設定し、開始します。

NYSE の MAMA と MAMDA のクライアント API を実装する Data Source Handler は、NYSE データ・フィードに接続し、セッションを開始し、データ辞書をダウンロードします。 SpSubscriber と SpPublisher のコンポーネントが、SDK インタフェースを介して Event Stream Processor に接続します。 SpSubscriber はウォッチリストの受信を開始し、SpPublisher はデータをデータ・ストリームにパブリッシュする準備を整えます。

アダプタの実行中のインスタンスが存在するときに **start** コマンドを実行すると、このコマンドは無視され、警告が送信されます。

参照：

- *NYSE アダプタの起動* (257 ページ)

stop コマンド

stop コマンドは、SpPublisher と SpSubscriber を Event Stream Processor から切断します。これによって、Data Source Handler はセッションを閉じ、データソースから切断します。さらに、アダプタ・コントローラがユーザ・コマンドの受信を停止し、アダプタ・プロセスを終了します。

アダプタの実行中のインスタンスが存在しないときに **stop** コマンドを実行すると、コマンドは無視され、警告が送信されます。

参照：

- *NYSE アダプタの停止* (259 ページ)

status コマンド

status コマンドは、アダプタのステータスを報告します。アダプタ・コントローラが次のステータスを表示します。 実行中または停止中のいずれか。

参照：

- *NYSE アダプタのステータスの確認* (258 ページ)

ウォッチリスト

次の 2 つのウォッチリスト・ストリームを使用することによって、アダプタを介して、メッセージ・フローを動的に制御します。 マーケット・データとオーダー・ブック。

アダプタは、複数のトランスポートを介して、利用可能なネームスペースからさまざまな証券コードにサブスクライブします。 証券コード、ネームスペース、トランスポートの 3 つの組み合わせが、サブスクリプション・キーと呼ばれます。ウォッチリストは、サブスクリプション・キーをデータ・ストリームにマップします。ウォッチリスト・ストリームの名前は、アダプタ設定ファイルで定義されます。

アダプタは、マーケット・データのグループ・サブスクリプションをサポートします。 グループ・サブスクリプションは、グループ証券コードによって識別され、各グループ証券コードは、さまざまな個々の証券コードに関連付けられています。 同じグループの証券コードについて受信したデータは、同じデータ・ストリームに格納されます。 データ・ストリームのレコードは、個々の証券コードによってキー付けされます。 たとえば、証券コードの X1、X2、X3 が GROUPX に関連付けられていると、データ・レコードは、GROUPX ではなく X1、X2、X3 を使用してキー付けされます。

サブスクリプション・キーは、多対 1 としてストリームに関連付けられます。 同じように、サブスクリプション・キーは、1 つのマーケット・データと、売り側と買い側ごとに 1 つのオーダー・ブック・ストリームしかターゲットにできませ

ん。 ただし、データ・ストリームは、複数のサブスクリプション・キーのターゲットになれます。 売り側と買い側のオーダー・ブックは、同じストリームまたは別のストリームでホストできます。

ウォッチリストの挿入と削除はユーザ制御で、更新はエラー条件として解釈されます。 アダプタはウォッチリストの変更に対して次のように応答します。

挿入	指定されたトランスポート上の指定されたネームスペースの証券コードに対するサブスクリプションをアクティブ化する。
削除	サブスクリプションを非アクティブ化する。
更新	エラーをログ記録する。 設定に従って、警告をオペレータに送信します。 データを継続して古いストリームに送信します。ターゲット・ストリームを変更するには、アダプタをシャットダウンし、再起動します。

参照：

- *ウォッチリスト・ストリームの設定パラメータ* (251 ページ)
- *マーケット・データのウォッチリスト* (240 ページ)
- *オーダー・ブックのウォッチリスト* (241 ページ)
- *ウォッチリストのオペレーション* (259 ページ)
- *挿入* (259 ページ)
- *削除* (260 ページ)

マーケット・データのウォッチリスト

マーケット・データのウォッチリストの構造と内容の例を次に示します。

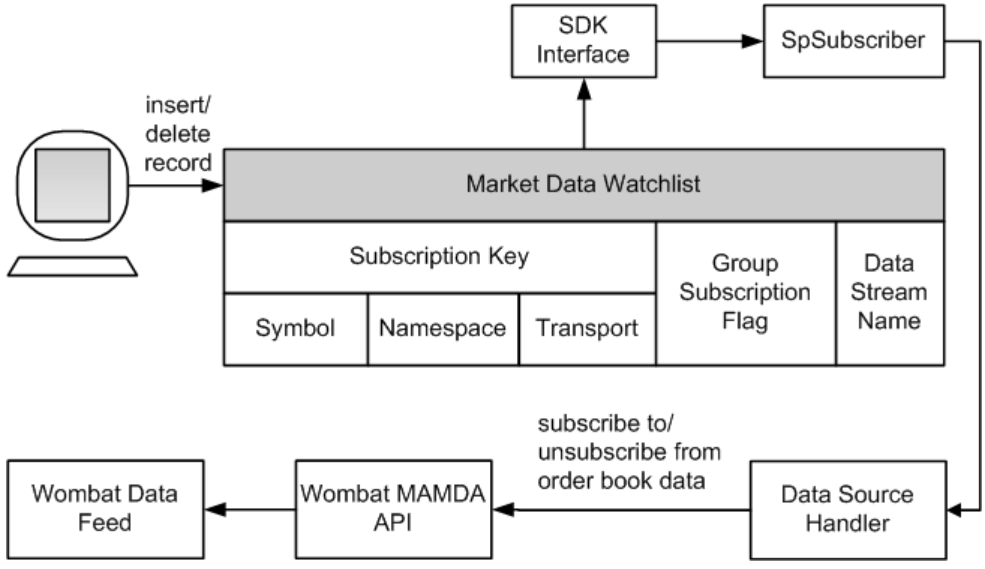


表 1 : マーケット・データのウォッチリストの構造と内容のサンプル

サブスクリプション・キー			グループ・サブスクリプション・フラグ	データ・ストリーム名
証券コード	ネームスペース	トランスポート		
BDK	NASDAQ	T1	false	MarketB
MSFT	NYSE	T1	false	MarketA
IBM	NASDAQ	T2	false	MarketA
GROUPX	NYSE	T2	true	MarketC

参照：

- *ウォッチリスト・ストリームの設定パラメータ* (251 ページ)
- *オーダー・ブックのウォッチリスト* (241 ページ)
- *ウォッチリスト* (239 ページ)

オーダー・ブックのウォッチリスト

オーダー・ブックのウォッチリストの構造と内容の例を次に示します。

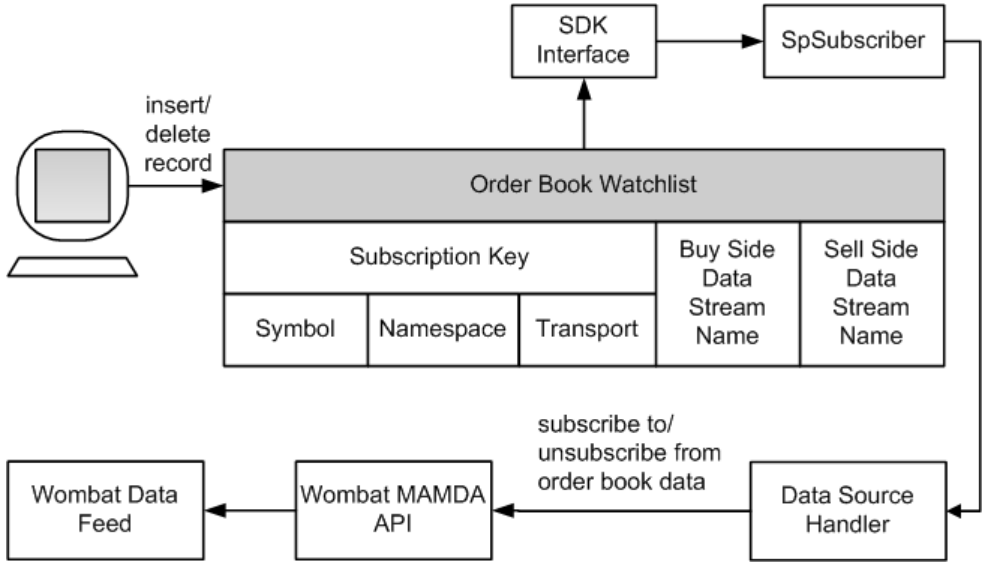


表 2 : オーダー・ブックのウォッチリストの構造と内容のサンプル

サブスクリプション・キー			売り側のデータ・ストリーム	買い側のデータ・ストリーム
証券コード	ネームスペース	トランスポート		
APPL	NASDAQ	T1	BookABuy	-
BDK	NASDAQ	T1	BookABuy	BookASell
MSFT	NYSE	T1	BookB	-
IBM	NASDAQ	T2	BookB	BookB

参照：
- *ウォッチリスト・ストリームの設定パラメータ* (251 ページ)
- *マーケット・データのウォッチリスト* (240 ページ)
- *ウォッチリスト* (239 ページ)

データ・ストリーム

データ・ストリームには、次の 2 つのタイプがあります。 マーケット・データとオーダー・ブック。

参照：
- *データ・ストリームの設定* (252 ページ)

マーケット・データのストリーム

マーケット・データのストリームには、レコード・キー、1 つ以上のフィールド、失効フラグがあります。

レコード・キーは、証券コード、ネームスペース (存在する場合)、トランスポート (存在する場合) で構成されます。 ネームスペース、トランスポート、またはその両方を省略すると、異なるネームスペースとトランスポートから受信し、同じ証券コードを持つ更新が、同じレコードに格納されます。

マーケット・データのストリームのカラムは、たとえば wBidSize や wBidPrice など、ホストされているフィールドと同じ名前を持つことがあります。 カスタマイズされた名前のカラム (たとえば、MyTimestamp) は、アダプタ設定ファイルのフィールド名にマップされます。

表 3 : マーケット・データのストリームの内容のサンプル

レコード・キー			wBidSize	wBidPrice	My Timestamp	失効
証券コード	ネームスペース	トランスポート				
MSFT	NYSE	T1	550	33.67	31--12--2008T 10:32:10.536	false
IBM	NASDAQ	T2	430	51.89	31--12--2008T 10:32:440.993	true
X1	NYSE	T2	850	133.63	31--12--2008T 10:27:580.563	false
X2	NYSE	T2	440	74.36	31--12--2008T 10:29:030.755	false
X3	NYSE	T2	180	21.53	31--12--2008T 10:31:55.001	false

参照：
- *データ・ストリームの設定* (252 ページ)

オーダー・ブックのデータ・ストリーム
オーダー・ブックのデータ・ストリームは、レコード・キー、エントリ ID、価格、総取引数、タイムスタンプ、失効フラグで構成されます。

価格レベルの数は無制限です。

レコード・キーは、証券コード、ネームスペース (存在する場合)、トランスポート (存在する場合)、サイド・インジケータ (存在する場合)、価格で構成されます。 ネームスペース、トランスポート、またはその両方を省略すると、異なるネームスペースとトランスポートから受信した、同じ証券コードを持つ更新は統合されます。

サイド・インジケータの有効な値は、B (売り側) と A (買い側) です。 サイド・インジケータのカラムは、データ・ストリームがオーダー・ブックの両側のターゲットになっている場合は必須です。 サブスクリプションでターゲットの売りと買いのストリームが異なる場合、サイド・インジケータのカラムの値は、アダプタによって無視されます。

表 4 : オーダー・ブックのストリームの内容のサンプル

レコード・キー				エントリ ID	サイズ	タイムスタンプ	失効
証券コード	ネームスペース	売り側／買い側	価格				
IBM	NASDAQ	B	106.23	25345	200	31--12--2008 T10:32:10.536	false
IBM	NASDAQ	B	106.22	74558	300	31--12--2008 T10:32:09.211	false
IBM	NASDAQ	B	106.19	12347	600	31--12--2008 T10:32:07.840	false
IBM	NASDAQ	A	108.73	53298	200	31--12--2008 T10:32:05.266	false
IBM	NASDAQ	A	108.11	53749	300	31--12--2008 T10:32:03.754	false
MSFT	NYSE	-	55.93	65228	400	31--12--2008 T10:31:53.922	false
MSFT	NYSE	-	55.87	54349	700	31--12--2008 T10:31:46.725	false

注意： トランスポートはデータ・レコードの格納時に無視されます。 また、サイド・インジケータは、その値がウォッチリストから抽出される場合は空です。

参照：

- *データ・ストリームの設定* (252 ページ)

失効レコード

データ・ストリームのレコードは、特定のアダプタで処理が失敗すると、「失効」とマーク付けされます。

アダプタは起動すると、終了命令を Event Stream Processor に送信します。 これらの命令は、アダプタとの通信が失われるか、アダプタがハートビートの送信を連続して 2 回失敗すると実行されます。 終了プロシージャは、設定されているすべてのアダプタ・データ・ストリームのすべてのレコードの失効フラグを 1 (true) に設定します。

データ・ストリームのレコードは、次の場合にも「失効」とマーク付けされます。

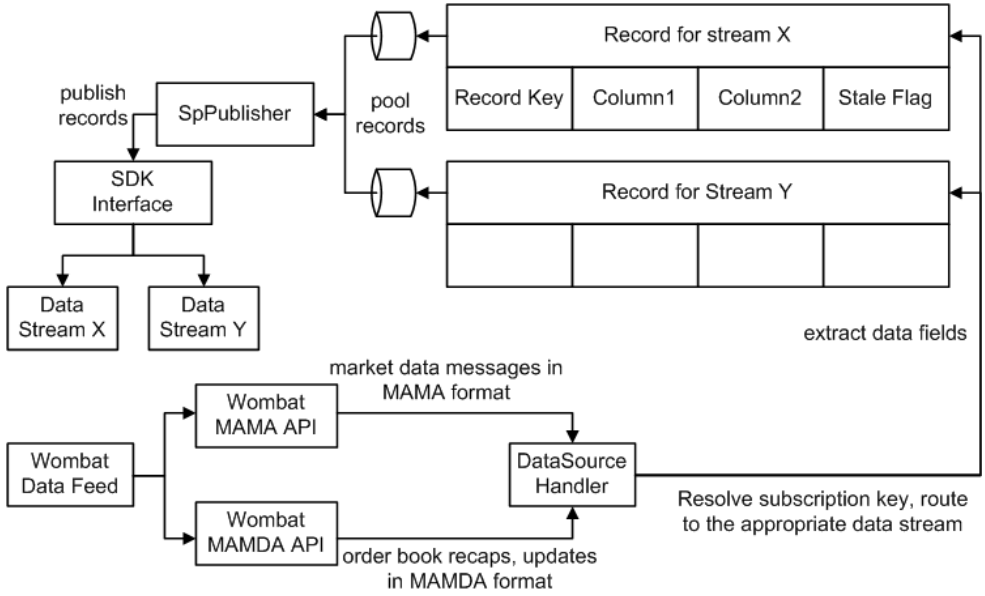
- サブスクリプションが非アクティブ化された。
- アダプタがマーケット・データのギャップ・メッセージを受信した。
- アダプタがオーダー・ブックのギャップ・メッセージを受信した。
- アダプタが NYSE データ・フィードからトランスポート切断メッセージを受信した。
- ユーザ・コマンドまたは致命的なエラー条件によって、アダプタがシャットダウン処理を開始した。

参照：

- *Event Stream Processor のパラメータ* (249 ページ)

メッセージ・フロー

アダプタのメッセージ・フローは、start コマンドによって開始されます。



Data Source Handler は、マーケット・データのメッセージを MAMA フォーマットでリアルタイムに受信します。 オーダー・ブックのリキャップと更新 (デルタや複合デルタなど) を MAMDA フォーマットで受信します。 また、サブスクリプション・キーを解決し、メッセージを該当するデータ・ストリームに転送します。転送先では、データ・フィールドからストリーム・レコードへの抽出と変換が実行されます。

レコードは Event Stream Processor にパブリッシュするために準備されますが、すぐにはパブリッシュされません。レコードはキューに登録され、SpPublisher によって取り出されます。 キューの容量は、アダプタ設定ファイルで設定します。キューのサイズを大きくすると、突発的に大量のメッセージが発生してもオーバ

フローが起こりにくくなります。 キューに登録されているレコードがキューの容量の 4 分の 3 を超えると、警告がログ記録されます。容量が 4 分の 3 未満に戻ると、別の警告がログ記録されます。 キューがいっぱいになると、次のレコードを登録できるようになるまで、アダプタは待機します。

アダプタはパフォーマンスを向上させるために、レコード・プーリングを使用します。 キューから取り出されたレコードは、アダプタ設定ファイルのユーザ設定に基づいてプールされます。 メッセージがパブリッシュされるのは、プール・サイズが最大プール・サイズに到達した、または最後にパブリッシュしてから最大プーリング時間が経過した、のいずれか早い方が発生したときです。 最大プール時間は、アダプタの遅延に影響します。 最大プール時間があまりに短いか、最大プール・サイズがあまりに小さいと、メッセージが小さなバッチで Event Stream Processor にパブリッシュされ、全体的なパフォーマンスが劣化します。

プールされているレコード・バッチをパブリッシュする準備が整うと、SpPublisher は Pub/Sub API を使用して、レコードを Event Stream Processor に送信します。 レコードは非同期にパブリッシュされます。 アダプタは、いかなるフィードバックも Event Stream Processor から受信しません。 フェールオーバが発生すると、Pub/Sub API はメッセージを失うことなく、パブリッシュ先を予備の Event Stream Processor インスタンスに切り替えます。

NYSE アダプタのデータ型のマッピング

Event Stream Processor のデータ型は、NYSE のデータ型にマップされます。

Event Stream Processor のデータ型	MAMA API のデータ型
`integer`	`long`
`long`	`long`
`float, money, money1-money15`	`double`
`string`	`string`
`bigdatetime`	`com.wombat.mama.MamaDateTime`
`interval`	`com.wombat.mama.MamaDateTime`
`binary`	`string`

JAVA_HOME 環境変数の設定
Java ディレクトリを指すように JAVA_HOME 環境変数を設定します。

前提条件
Java Runtime Environment のバージョン 1.6.0_26 以降をインストールする。 NYSE Java とバイナリ・ライブラリを `$ESP_HOME/adapters/wombat/lib/wombat` の下に配置します。

手順
JAVA_HOME 環境変数を、Java Runtime Environment 1.6.0_26 以降がインストールされているディレクトリ・パスに設定します。

次のステップ
ESP_HOME 環境変数が正しく設定されていることを確認する。

設定
NYSE Technologies アダプタ用の設定情報を以下に示します。

NYSE アダプタのディレクトリ
アダプタのディレクトリには、設定ファイル、テンプレート、例、JAR ファイルなど、アダプタに関連するすべてのファイルがあります。

```
README.txt  Quick Guide
ReleaseNotes.txt  Release Notes

bin/
    adapter.bat  Standalone adapter startup script
    adapter.sh      Standalone adapter startup script
    adapter-plugin.bat Plug-in connector startup script
    adapter-plugin.sh  Plug-in connector startup script

config/
    controller.xsd     Controller schema
    log4j.properties   Sample logging configuration
    wombatadapter.xsd  Adapter schema
    dictionary.txt     The configuration file to map wombat fields
with ESP datatypes. This is used for data dictionary.
    login.config  Authentication configuration

discovery/        Data discovery templates

example/        Working example

lib/
    esp_wombat_adapter.jar  wombat adapter library
    wombat/        wombat java and binary libraries

javadoc/
```

```
    adapterapi/     Adapter API Javadoc
    wombatadapter/     Wombat Adapter Javadoc

Common jars are located:

$ESP_HOME/adapters/jar

activation.jar                Java mail library
adapterapi.jar                Adapter API code
axis.jar                     Webservices jar
commons-codec-1.3.jar         Required by SDK API
commons-discovery-0.2.jar
commons-logging-1.1.jar    Logging library
esp_java_sdk-0.2.jar     ESP SDK library
jaxrpc-api-1.1.jar               Required by ESP SDK
log4j-1.2.14.jar          Logging library
mail.jar              Java mail library
saaj-api-1.3.jar              Webservices jar
ws-commons-util-1.0.2.jar  Required by ESP SDK
wsdl4j-1.5.1.jar
xercesImpl.jar         XML parser library
xmlrpc-client-3.1.3.jar Required by ESP SDK
xmlrpc-common-3.1.3.jar Required by ESP SDK
xmlrpc-server-3.1.3.jar Required by ESP SDK
```

スキーマと設定ファイル

アダプタ設定はファイルからロードされ、アダプタのスキーマに対して検証されます。

実際に動作するアダプタの例が、サンプル `adapter.xml` ファイルにあります。このファイルを編集するか、新しいファイルを作成できます。 アダプタ設定がスキーマに対して正しく検証されることを確認します。 設定が正しく検証されないと、エラー・メッセージが表示されます。

アダプタ・コントローラ・パラメータ

controllerPort パラメータは、アダプタの「コマンドと制御」のポートを指定します。

パラメータ名	タイプ	説明
controllerPort	`positive integer`	(必須) アダプタの「コマンドと制御」のポートを指定。 ユーザ・コマンドは、localhost 上のこのポートに送信されます。

Event Stream Processor のパラメータ

Event Stream Processor のパラメータは、Event Stream Processor と NYSE アダプタとの間の通信を設定します。

これらのパラメータは、`controller.xsd` ファイル (`config` ディレクトリ内) で定義されます。

パラメータ名	タイプ	説明
espAuthType	`string`	(必須) Event Stream Processor への認証に使用される方法を指定。 有効な値は、以下のとおりです。 • **server_rsa –** キーストアを使用する RSA 認証 • **user_password –** Kerberos 認証と LDAP 認証 • **none –** 認証なし アダプタがスタジオ・プラグインとして動作している場合、**espAuthType** は **Authentication Mode** スタジオ起動パラメータで上書きされます。
espUser	`string`	(必須) Event Stream Processor にログインするために必要なユーザ名を指定。 none (**espAuthType** を参照してください) 以外のすべての認証スキームで必要です。 デフォルト値はありません。
espPassword	`string`	(必須) Event Stream Processor にログインするために必要なパスワードを指定。 none (**espAuthType** を参照してください) 以外のすべての認証スキームで必要です。 **espPassword** 値が暗号化されているかどうかを示す "encrypted" 属性も指定します。デフォルト値は false です。 true に設定すると、パスワード値は **espRSAKeyStore** と **espRSAKeyStorePassword** を使用して復号化されます。

パラメータ名	タイプ	説明
espProjectUri	`string`	(必須) Event Stream Processor クラスタに接続するための完全なプロジェクト URI を指定。たとえば、`esp://localhost:19011/ws1/p1` のように記述します。
pulseInterval	`non-negative integer`	(必須) 期間を秒単位で指定。この期間では、アウトバウンド・レコードの変更は Event Stream Processor によって結合され、アダプタによって単一のイベントとして受信されるようになります。 設定されていないか、0 に設定されると、レコードの変更は、発生するごとに個々に受信されます。
espHeartbeatPeriod	`positive integer`	(オプション) 秒数を指定。この秒数が経過すると、アダプタは次のハートビートを Event Stream Processor に送信します。 Event Stream Processor が連続して 2 回のハートビート受信に失敗すると、アダプタがパブリッシュしたすべてのレコードが失効とマーク付けされます。 デフォルト値は 10 です。
recordQueueCapacity	`positive integer`	(オプション) レコード・キューの容量を指定。デフォルト値は、4096 です。
maxPubPoolSize	`positive integer`	(オプション) レコード・プールの最大サイズを指定。 レコード・プーリングは、パブリッシュ処理の高速化に役立ちます。デフォルト値は 256 です。
maxPubPoolTime	`positive integer`	(オプション) レコードがパブリッシュされるまでにプールに存在可能な最大時間をミリ秒単位で指定。 設定されていない場合、プーリング時間は無制限で、プーリング方式は **maxPubPoolSize** によって制御されます。 デフォルト値はありません。

パラメータ名	タイプ	説明
useTransactions	`boolean`	(オプション) true に設定すると、プールされたメッセージがトランザクションで Event Stream Processor にパブリッシュされる。 false に設定すると、エンベロープでパブリッシュされます。デフォルト値は false です。
espRSAKeyStore	`string`	(場合に応じて必須) RSA キーストアのロケーションを指定し、パスワード値を復号化するために使用される。 **espAuthType** が server_rsa に設定されている、**espPassword** の暗号化属性が true に設定されている、または両方の場合に必要です。
espRSAKeyStorePassword	`string`	(場合に応じて必須) キーストア・パスワードを指定し、パスワード値を復号化するのに使用される。**espAuthType** が server_rsa に設定されている、**espPassword** の暗号化属性が true に設定されている、または両方の場合に必要です。
espEncryptionAlgorithm	`string`	(オプション) **espPassword** の暗号化属性が true に設定されると使用される。 ブランクのままにすると、RSA がデフォルトとして使用されます。

参照：
- *失効レコード* (244 ページ)

ウォッチリスト・ストリームの設定パラメータ
ウォッチリスト・ストリームの設定パラメータは、マーケット・データとオーダー・ブックのウォッチリストの名前を指定します。

パラメータ名	タイプ	説明
marketDataWatchlist	`string`	(必須) マーケット・データのウォッチリストの名前を指定する。
orderBookWatchlist	`string`	(必須) オーダー・ブックのウォッチリストの名前を指定する。

参照：
- *マーケット・データのウォッチリスト* (240 ページ)

- *オーダー・ブックのウォッチリスト* (241 ページ)
- *ウォッチリスト* (239 ページ)

データ・ストリームの設定

設定ファイルの `marketDataStreams` セクションを使用して、データ・ストリームのパラメータを提供します。

各データ・ストリームのストリーム名を示します。

データ・ストリームのカラムと対応する MAMA フィールドは、名前が同じ場合と異なる場合があります。 名前が異なる場合、カラムと対応するデータ・フィールドを明示的にマップします。 次の例では、MyTimestamp カラムが wSrcTime MAMA フィールドにマップされています。

```
<column>
<name>MyTimestamp</name>
<field>wSrcTime</field>
</column>
```

カラムと対応するフィールドが同じデータ型であることを確認します。 一部のカラムに対して対応するフィールドがない場合があります。カラム名の Symbol、Namespace、Transport、Stale は予約語です。

参照：

- *データ・ストリーム* (242 ページ)
- *マーケット・データのストリーム* (242 ページ)
- *オーダー・ブックのデータ・ストリーム* (243 ページ)

データフィードのパラメータ

データフィードのパラメータは、NYSE アダプタ用のデータフィードを設定します。

これらのパラメータの詳細については、『MAMA Developer's Guide』を参照してください。

パラメータ名	タイプ	説明
middleware	`string`	(必須) ミドルウェア API の名前を指定。 現在、Configuration 9 wmw (NYSE TCP Middleware) のみがサポートされています。 デフォルト値は、wmv または lbm です。

パラメータ名	タイプ	説明
subscriptionTimeout	`positive integer`	(必須) アダプタが待機する秒数を指定。初期値を受信せずにこの時間が経過すると、マーケット・データのサブスクリプション要求を再送信します。
subscriptionRetries	`positive integer`	(必須) サブスクリプションの初期イメージを取得するための再試行の回数を指定。
dictionaryTransport	`string`	(必須) アダプタのスタートアップ時に MAMA 辞書を要求するトランスポートを指定。
dictionaryNamespace	`string`	(必須) アダプタのスタートアップ時に MAMA 辞書を要求するネームスペースを指定。
dictionaryTimeout	`positive integer`	(必須) アダプタが待機する秒数を指定。応答を受信せずにこの時間が経過すると、MAMA 辞書要求を再送信します。
dictionaryRetries	`positive integer`	(必須) MAMA 辞書を取得するための再試行の回数を指定。

NYSE 設定ファイルのサンプル

NYSE アダプタ用のサンプル設定ファイル (`adapter.xml`) を以下に示します。

このファイルは、`example` にあります。

```
<adapter>

<!-- Adapter Controller -->
 <controller>
  <controllerPort>13579</controllerPort>
 </controller>

<!-- Streaming platform settings -->
 <esp>
   <espConnection>
        <espProjectUri>esp://localhost:19011/w1/p1</espProjectUri>
   </espConnection>

   <espSecurity>
        <espUser>espuser</espUser>
        <espPassword encrypted="false">espuser</espPassword>
        <espAuthType>none</espAuthType>
<!--
        <espRSAKeyStore>/keystore/keystore.jks</espRSAKeyStore>
       <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword> --
>
        <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
```

```
   </espSecurity>
  <maxPubPoolSize>1</maxPubPoolSize>
 </esp>

 <watchlists>
  <marketDataWatchlist>MarketDataWatchlist</marketDataWatchlist>
  <orderBookWatchlist>OrderBookWatchlist</orderBookWatchlist>
 </watchlists>

 <marketDataStreams>
  <stream>
   <name>MyMarketDataStream</name>
   <column>
    <name>MyTimestamp</name>
    <field>wSrcTime</field>
   </column>
  </stream>
 </marketDataStreams>

<datafeed>
 <middleware>wmw</middleware>
 <subscriptionTimeout>5</subscriptionTimeout>
 <subscriptionRetries>1</subscriptionRetries>
 <dictionaryTransport>demo</dictionaryTransport>
 <dictionaryNamespace>WOMBAT</dictionaryNamespace>
 <dictionaryTimeout>10</dictionaryTimeout>
 <dictionaryRetries>1</dictionaryRetries>
</datafeed>

</adapter>
```

NYSE インプット・アダプタ

NYSE インプット・アダプタは、NYSE データ・フィードに接続し、リアルタイムのレベル 1 と 2 のデータを受信します。

任意のソース・ストリーム上のアダプタを入力データ・ロケーションとして設定できます。 認証方法は、次の 3 つの中から Event Stream Processor と同じに設定されます。 none、rsa、または gssapi。 このアダプタは、スキーマ検出をサポートします。

このアダプタを使用するには、NYSE アダプタのバージョン 1 以降がインストールされている必要があります。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Connector Directory Path	**baseDir**	`directory`	(必須) アダプタ・インストール・ディレクトリへのパスを指定。このプロパティは、「コネクタ・リモート・ディレクトリ・パス」プロパティが指定されていると無視されます。 デフォルト値は、`/mydir/NYSEAdapter` です。
Configuration File Path	**configFilePath**	`configFilename`	(必須) アダプタ設定ファイルの絶対パスを指定。 このプロパティは、**Remote Configuration File Path** プロパティが指定されていると無視されます。 デフォルト値は、`/mydir/NYSEAdapter/config/adapter.xml` です。
Discovery Directory Path	**discDirPath**	`directory`	(必須) アダプタ検出ディレクトリの絶対パスを指定。 デフォルト値は、`/mydir/NYSEAdapter/discovery` です。
Connector Remote Directory Path	**remoteBaseDir**	`string`	(詳細) アダプタ・リモート・ベース・ディレクトリのパスを指定 (リモート実行のみ)。 このプロパティが指定されている場合、**Connector Directory Path** プロパティは無視されます。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Remote Configuration File Path	**remoteConfigFilePath**	`string`	(詳細) アダプタ設定ファイルの絶対パスを指定 (リモート実行のみ)。 このプロパティが指定されている場合、**Configuration File Path** プロパティは無視されます。
Discovered Table (runtime)	**table**	`tables`	(詳細) 検出されたテーブルの名前。 これは、スタジオによって設定されます。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

ロギング

NYSE アダプタは Apache `log4j` API を使用して、エラー、警告、情報、デバッグ・メッセージをログ記録します。

`log4j.properties` ファイルには、ログ記録設定があります。 このファイルのサンプルが、アダプタ配布にあります。

注意： ログ記録レベルを DEBUG に設定すると、ログ・ファイルが非常に大きくなることがあります。 デフォルトのレベルは、INFO です。 ロー Wombat メッセージが、DEBUG レベルでログ記録されます。

ログ記録設定の詳細については、http://logging.apache.org/log4j を参照してください。

オペレーション

NYSE アダプタをコマンド・ラインから操作します。

実行するプロジェクトには、マーケット・データとオーダー・ブックのウォッチリストがあることを確認します。 ウォッチリスト・ストリームの名前は、それぞれ **marketDataWatchlist** パラメータと **orderBookWatchlist** パラメータに対応していることを確認します。

該当するロギング・レベルを `log4j.properties` で設定します。

NYSE アダプタの起動

NYSE アダプタをコマンド・ラインから起動するには、Event Stream Processor を起動して、**start** コマンドを実行します。

1. Event Stream Processor を起動します。

   Windows：

   1. サンプル・クラスタを起動します。

      ```
      cd %ESP_HOME%¥cluster¥nodes¥node1
        %ESP_HOME%¥bin¥esp_server --cluster-node node1.xml
      ```

   2. CCL をコンパイルして CCX を作成します。

      ```
      %ESP_HOME%¥bin¥esp_compiler -i model.ccl -o model.ccx
      ```

   3. プロジェクトをクラスタに展開します。

      ```
      %ESP_HOME%¥bin¥esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --add_project --workspace-
      name=w1 --project-name=p1 --ccx=model.ccx
      ```

   4. 展開したクラスタをサーバで起動します。

      ```
      %ESP_HOME%¥bin¥esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --start_project --
      workspace-name=w1 --project-name=p1
      ```

   UNIX：

   1. サンプル・クラスタを起動します。

      ```
      cd $ESP_HOME/cluster/nodes/node1
        $ESP_HOME/bin/esp_server --cluster-node node1.xml
      ```

   2. CCL をコンパイルして CCX を作成します。

      ```
      $ESP_HOME/bin/esp_compiler -i model.ccl -o model.ccx
      ```

   3. プロジェクトをクラスタに展開します。

      ```
      $ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --add_project --workspace-
      name=w1 --project-name=p1 --ccx=model.ccx
      ```

   4. 展開したクラスタをサーバで起動します。

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --start_project --
workspace-name=w1 --project-name=p1
```

2. アダプタを起動します。

オペレーティング・システム	手順

UNIX	端末ウィンドウを開き、次を入力。
	`cd $ESP_HOME/adapters/wombat/bin`
	`./adapter.sh <configuration file path> start`

Windows	コマンド・ウィンドウを開き、次を入力。
	`cd %ESP_HOME%/adapters/wombat/bin`
	`adapter.bat <configuration file path> start`

esp_subscribe ユーティリティを使用して、NYSE メッセージが正しく Event Stream Processor にパブリッシュされることを確認できます。

参照：

- *start コマンド* (238 ページ)

NYSE アダプタのステータスの確認

NYSE アダプタのステータスをコマンド・ラインから確認するには、**status** コマンドを実行します。

オペレーティング・システム	手順

UNIX	端末ウィンドウを開き、次を入力。
	`cd $ESP_HOME/adapters/wombat/bin`
	`./adapter.sh <configuration file path> status`

Windows	コマンド・ウィンドウを開き、次を入力。
	`cd %ESP_HOME%/adapters/wombat/bin`
	`adapter.bat <configuration file path> status`

アダプタのステータスが、実行中または停止中のいずれかとして表示されます。

参照：
- *status コマンド* (239 ページ)

NYSE アダプタの停止
NYSE アダプタをコマンド・ラインから停止するには、**stop** コマンドを実行します。

オペレーティング・システム	手順

UNIX	端末ウィンドウを開き、次を入力。
	`cd $ESP_HOME/adapters/wombat/bin`
	`./adapter.sh <configuration file path> stop`

Windows	コマンド・ウィンドウを開き、次を入力。
	`cd %ESP_HOME%/adapters/wombat/bin`
	`adapter.bat <configuration file path> stop`

参照：
- *stop コマンド* (239 ページ)

ウォッチリストのオペレーション
ウォッチリストは、挿入と削除を使用して変更できます。

ウォッチリストの更新は、エラー条件として解釈され、アクションは何も実行されません。

マーケット・データのウォッチリストを変更すると、アダプタは証券コードのリアルタイム・データにサブスクライブするか、またはサブスクリプションを削除します。オーダー・ブックのウォッチリストを変更すると、アダプタは証券コードのオーダー・ブック・データにサブスクライブするか、またはサブスクリプションを削除します。

参照：
- *ウォッチリスト* (239 ページ)

挿入
ウォッチリストの挿入は、アダプタで次の 2 つのアクションをトリガします。 サブスクライブとパブリッシュ。

ウォッチリストを挿入すると、次のアクションがトリガされます。

- アダプタは、指定されたネームスペースからの指定された証券コードに、指定されたトランスポートを介してサブスクライブする。
- アダプタは、リアルタイムのマーケット・データ・メッセージまたはオーダー・ブックのリキャップと更新を受信し、それらを対応するデータ・ストリームにパブリッシュする。

参照：
- *ウォッチリスト* (239 ページ)

削除
ウォッチリストの削除は、アダプタで次の 2 つのアクションをトリガします。 サブスクリプションの削除とレコードの失効化。

ウォッチリストを削除すると、次のアクションがトリガされます。

- アダプタは、指定されたトランスポート経由の指定されたネームスペースの指定された証券コードのサブスクリプションを削除する。
- キャンセルされたサブスクリプションからの結果となるマーケット・データ・ストリームのレコードを失効化する。

参照：
- *ウォッチリスト* (239 ページ)

例： データへのサブスクライブとデータのパブリッシュ
2 つの証券コードのリアルタイム・マーケット・データと 1 つの証券コードのオーダー・ブック・データにサブスクライブし、受信データを Event Stream Processor にパブリッシュします。

前提条件
NYSE データフィードへのネットワーク接続が必要です。

手順

1. `adapter.sh` スクリプトを編集します。
2. JAVA_HOME 環境変数を、Java Runtime Environment (JRE) がインストールされているディレクトリに設定します。

   注意： NYSE ライブラリは、32 ビットと 64 ビットの両方のバージョンで利用できます。 ライブラリが 32 ビットの場合は、32 ビット JRE を使用します。 ライブラリが 64 ビットの場合は、64 ビット JRE を使用します。
3. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`

Windows	コマンド・ウィンドウを開く。
	1. サンプル・クラスタを起動。
	`start_server_cluster.bat`
	2. プロジェクトをクラスタに追加し、クラスタ上で起動。
	`start_project.bat`

4. アダプタの `lib/wombat` ディレクトリの `mama.properties` ファイルを編集して、`subscribe_address` プロパティと `subscribe_port` プロパティが NYSE データ・フィードを指し示すようにします。
5. アダプタを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./adapter.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `adapter.bat`

6. アダプタの初期化が完了するまで 5 ～ 10 秒待機します。
7. ストリーム・レコードをアップロードします。

オペレーティング・システム	手順
UNIX	`./upload.sh`
Windows	`upload.bat`

8. サブスクライバ・ユーティリティを起動して、データ・ストリームの内容を表示します。

オペレーティング・システム	手順
UNIX	`./esp-subscribe.sh`
Windows	`esp-subscribe.bat`

オープン・アダプタ

Sybase Event Stream Processor のオープン・アダプタは、オープンソースの openadapter™ (openadapter.org) の 1 つのバージョンです。

Web サービスやさまざまなファイル・タイプなど、一般的なアプリケーションやミドルウェア環境用に、多くのアダプタが提供されています。 各アダプタをさまざまなリーダやライタと共に使用することによって、さまざまなタイプのメッセージ (たとえば、区切られたフィールド・レコードや XML ドキュメント) を解析しフォーマットできます。 アダプタを通して受信されるレコードには、ESP_OPS カラムがあり、そのレコードに対して実行するデータベース・オペレーションを示します。

- i、I、insert、または INSERT は挿入を示す。
- p、P、upsert、まはた UPSERT はアップサートを示す。
- u、U、update、または UPDATE は更新を示す。
- s、S、safedelete、または SAFEDELETE は安全な削除を示す。
- d、D、delete、または DELETE は削除を示す。

ESP_OPS カラムを使用する場合、各レコードのこのカラムに値があることを確認します。

オープン・アダプタは、アダプタ・プロパティ・ファイルによって定義され、1 つ以上のソース・コンポーネントから 1 つ以上のシンク・コンポーネントにデータを移動する複数のコンポーネントによって構成されます。 中間コンポーネント (パイプ) を設定して、データに対して追加処理も実行できます。 多くのアダプタが実行できるシステムでは、各アダプタは、個別に起動と制御を行える個別のインスタンスとして実行します。

注意： Microsoft Windows では、二重の円記号 ¥¥ をパス、クラス・パス、URL の区切り文字として使用します。

オープン・アダプタを使用するには、個別のライセンスを購入する必要があります。このライセンスは、Sybase 製品ダウンロード・サイトから取得できます。 標準の SySAM 猶予期間がサポートされます。つまり、ライセンスなしで 30 日間実行できます。 この期間が過ぎた場合、実行するには有効なライセンスが必要です。

オープン・アダプタのデータ型のマッピング

Event Stream Processor のデータ型は、オープン・アダプタのデータ型にマップされます。

Event Stream Processor のデータ型	オープン・アダプタのデータ型
integer	integer
long	long
float	double
date	datetime
timestamp	datetime
bigdatetime	long
boolean	short
interval	long
binary	string
money	double
money(1-n)	double
string	string

JAVA_HOME 環境変数の設定

Java ディレクトリを指すように JAVA_HOME 環境変数を設定します。

JAVA_HOME 環境変数を、Java Runtime Environment 1.6.0_26 以降がインストールされているディレクトリ・パスに設定します。

次のステップ

ESP_HOME 環境変数が正しく設定されていることを確認する。

設定

アダプタ・プロパティ (`.props`) ファイルはテキスト・ファイルで、アダプタ用に呼び出されるコンポーネントの設定情報で構成されます。

プロパティ・ファイルは、テキスト・エディタまたはアダプタ・フレームワーク・エディタを使用して作成できます。 設定は、ソース、シンク、パイプの各コンポーネント、それらのリーダとフォーマッタで構成されます。 Sybase オープン・アダプタは、プロパティを XML ドキュメントからも読み込めます。

各アダプタ・プロパティ・ファイルには、単一のアダプタの設定があります。 アダプタ・プロパティは、アダプタ、コンポーネント、プロパティのそれぞれの名前を指定します。

```
AdapterName. ComponentName.PropertyName=PropertyValue
```

例を示します。

最初のカラムの # 文字は、コメント行を示します。

```
#
# Adaptor ‘Dynamic2’ Component ‘BalanceInMQ
#
Dynamic2.BalanceInMQ.QueueManager=QM_Test
Dynamic2.BalanceInMQ.Queue=TEST.BALANCE.IN … #
# Adaptor ‘Dynamic2’ Component ‘EventInMQ’
#
Dynamic2.EventInMQ.QueueManager=QM_Tes t
Dynamic2.EventInMQ.Queue=TEST.EVENT.IN
```

プロパティは、複数のレベルで修飾できます。 フィールドを定義するプロパティには、フィールド名とフィールドの型の両方が定義されていることを確認します。 例を示します。

```
Dynamic2.EventInMQ.Field1=Date
Dynamic2.EventInMQ.Date.Name=CurrentDate Dynamic2.
EventInMQ.Date.Type=Datetime
```

オープン・アダプタは、個々のコンポーネントのプロパティをサポートします。 アダプタ・プロパティ・ファイルで多くの文を使用して、プロパティの定義を単純化できます。

オープン・アダプタのディレクトリ

アダプタのディレクトリには、設定ファイル、テンプレート、例、JAR ファイルなど、アダプタに関連するすべてのファイルがあります。

```
$ESP_HOME/adapters/esp_open  Root directory for the Open adapter.
This directory contains the log4j.xml configuration file

    lib/ All adapter and third-party distributable jar files.

    bin/ Example scrips for starting the adapters and editor.

    repo/ Standard location for all property files.

    examples/ Various component examples.
```

ファイルをインクルードするための構文

その他のプロパティ・ファイルをインクルードするための構文。

```
#include other.props
```

ファイル名には、プレフィックスを追加できます。このプレフィックスは、インクルードされるファイル内の各プロパティ名に付加されます。

```
#include A.comp other.props
```

ここで、other.props には以下が指定されています。

```
property1=foo
```

オープン・アダプタは、次を読み込みます。

```
A.comp.property1=foo
```

変数の代入

プロパティ・ファイル内またはインクルードされるプロパティ・ファイル内で変数を定義します。

変数は、プロパティ・ファイル内またはインクルードされるプロパティ・ファイル内で定義できます。変数は、定義した後に使用できます。

```
NUM_TO_SEND=1000 … A.comp.MaxRecords=${NUM_TO_SEND}
```

また、変数の定義は、アダプタの起動時に、"-D" オプションを Java 仮想マシンに対して使用することによってもできます。

```
java –DNUM_TO_SEND=1000 org.openadapter.adapter.RunAdapter
config.props A
```

プロパティ名でのワイルドカード文字の使用

ワイルドカード名を使用して定義されたプロパティの場合、コンポーネントがプロパティを初期化して値を取得しようとすると、SuperProperties クラスは、要求に一致する、さらに特定されるプロパティ設定がないかぎり、ワイルドカード・プロパティの値を返します。

たとえば、次は、任意のアダプタとコンポーネントの名前に一致します。

```
*.QueueManager=QM_Test
```

次は、ワイルドカード文字部分が任意の 1 文字のコンポーネントに一致します。

```
A.?.QueueManager=QM_Test
```

A.B.QueueManager は一致しますが、A.B.C.QueueManager は一致しません。

自動インクリメントされるプロパティ名

大量のプロパティを指定する必要がある場合は、自動インクリメントされるプロパティ名を選択できます。

例を示します。

```
A.comp.field1=foo A.comp.field2=bar A.comp.field3=hello
A.comp.field4=world
```

次のフィールドを自動インクリメントできます。

```
A.comp.field++=foo A.comp.field++=bar A.comp.field++=hello
A.comp.field++=world
```

注意： アダプタ・フレームワーク・エディタを使用する場合、自動インクリメント・フィールドは、設定ファイルの読み込み時に、対応する番号付きフィールドに変換されます。 アダプタ・フレームワーク・エディタは、プロパティ・ファイルを書き戻すときにフィールドを自動インクリメント・フィールドに戻せません。

XML プロパティ・ファイル

アダプタ・プロパティ・ファイルを XML ドキュメントとして指定します。

```
<?xml version=”1.0” encoding=”UTF-8” ?>
<openadapter>
<A>
Component1>
<Name>C1</Name>
</Component1>
<Component2>
<Name>C2</Name>
</Component2>
<C1>
<ClassName>com.aleri.adapter.ibm.MqSource</ClassName>
</C1>
<C2>
…
…
</A>
</openadapter>
```

オープン・アダプタのコンポーネント

オープン・アダプタを使用するには、少なくとも 1 つのソース・コンポーネントと 1 つのシンク・コンポーネントを追加します。 ソース・コンポーネントは提供されたデータを読み込み、シンク・コンポーネントは、関連付けられた出力に書き込みます。

各コンポーネントには、固有の必須プロパティがあります。 `DOStringReader` プロパティと `DOStringWriter` プロパティを、ソース・コンポーネントとシンク・コンポーネントに設定すると、これらのコンポーネントを通るデータをさまざまなパーサで解析し、さまざまなフォーマットでフォーマットできます。

Event Stream Processor は、UTF-16 などのマルチバイト文字セットをサポートしません。しかし、外部ソースは非 ASCII 文字を指定できます。 デフォルトでは、アダプタは、非 ASCII 文字を 1 バイトまたは 2 バイトの Unicode 文字として解釈しますが、これによってデータが破損することがあります。 コードを明示的に設定するには、設定ファイルに `TextEncoding` プロパティを追加します。 例を示します。

```
Adapter. Component.TextEncoding=ASCII
```

複数のテーブル名を定義するプロパティが次のように指定されているとします。

```
Adapter. Component.Table++=TableName
```

この場合、設定ファイルでは次のように指定します。

```
Adapter. Component.Table1=TableName1
Adapter. Component.Table2=TableName2
```

プロパティは、複数のレベルをピリオドで区切って定義できます。 たとえば、Table1 に固有のプロパティは、次のように表せます。

```
Adapter. Component.TableName1.Field1=FieldName1
```

ソース・コンポーネント
オープン・アダプタには、2 つのソース・コンポーネントがあります。 AsapSource と SpPersistentSubscribeSource です。

AsapSource のプロパティ
AsapSource コンポーネントは、アダプタ設定で指定された名前の Event Stream Processor ストリームからのデータにサブスクライブします。

`ClassName=com.sybase.esp.adapter.asap.AsapSource`

プロパティ	説明
`ProjectUri`	(場合に応じて必須) クラスタ・モードで実行しているサーバに接続する (例：`esp://localost:19011/ws1/p1`)。
`User`	(必須) AsapSource と Event Stream Processor との間の最初の接続には、認証が必要。 Event Stream Processor に既知の有効なユーザ名を入力します。
`Passwd`	(必須) AsapSource と Event Stream Processor との間の最初の接続には、認証が必要。 上記で構成したユーザ名に対する有効なパスワードを入力します。 **注意：** `IsEncrypted` プロパティを true に設定すると、ユーザとパスワードの情報が Event Stream Processor に渡されてから、SSL 接続がセットアップされます。 これらの詳細は、プレーン・テキストで渡されます。
`IsEncrypted`	(オプション) このプロパティを true に設定すると、AsapSource は、Event Stream Processor に対して SSL ソケット接続を使用しようとする。

プロパティ	説明
`UseServerRSA`	(オプション) true を指定すると、サーバに接続するためにサーバ RSA 認証が使用される。 このプロパティを指定する場合、`KeyStore` プロパティと `KeyStorePassword` プロパティも指定する必要があります。 オープン・アダプタは、Bouncy Castle Java セキュリティ実装を使用します。 Bouncy Castle が、次の Java Runtime Environment ディレクトリの `java.security` ファイルに、他のセキュリティ・プロバイダと一緒にリストされていることを確認します。 `=org.bouncycastle.jce.provider.BouncyCastleProvider`
`KeyStore`	(オプション) サーバ RSA 認証に使用。 キーストア (.jks ファイル) のロケーションを指定します。 `UseServerRSA` を指定する場合は、このプロパティを設定します。
`KeyStorePassword`	(オプション) キーストアのパスワードを指定。 サーバ RSA 認証に使用されます。 `UseServerRSA` を指定する場合は、このプロパティを設定します。
`UseUserPassword`	(オプション) `User` と `Password` の認証 (たとえば、Kerberos) がサーバに接続するために使用される。 これを指定する場合、`User` と `Password` も指定する必要があります。
`Stream++`	(オプション) サブスクライブ先のストリームの名前。
`MaxBlockBuildTime (milliseconds)`	(オプション) AsapSource はパフォーマンスを向上させるためにレコードをブロックで送信。 ブロックには、`MaxBlockSize` に到達した、または `MaxBlockBuildTime` が経過した、のいずれか早い方が発生するまで、レコードが挿入されます。 ブロックは、アダプタ・パイプラインで送信されます。 0 に設定すると、ブロック構築時間に関する制限はありません。 デフォルト値は 1000 (ミリ秒) です。

プロパティ	説明
`MaxBlockSize`	(オプション) AsapSource はパフォーマンスを向上させるためにレコードをブロックで送信。 ブロックには、`MaxBlockSize` に到達した、または `MaxBlockBuildTime` が経過した、のいずれか早い方が発生するまで、レコードが挿入されます。 ブロックは、アダプタ・パイプラインで送信されます。 デフォルト値は 256 です。
`RecordBufferCapacity`	(オプション) 0 に設定すると、レコードはアダプタ・パイプラインで一度に 1 つずつ送信される。 正の数に設定すると、レコードがプラットフォームで使用可能な内部バッファのキューに入れられます。 バッファに入れられたレコードは専用スレッドの処理中のブロックに追加され、Pub/Sub サブスクリプション・スレッドは、レコードのバッファリングを継続できます。 バッファ容量を超えた場合、キューは、バッファ容量が再び使用可能になるまで抑制されます。 デフォルト値は 4096 です。
`PulseInterval`	(オプション) AsapSource が Event Stream Processor から次のレコードを取得するまで待機する秒数。 1 つのレコードに対してパルス間隔中にプラットフォームで生成される更新はすべて結合され、その結果生成されるレコードのみが送信されます。 デフォルトでは、パルス間隔での待機は実行されず、レコードは、アダプタによって即時に受信されます。

参照：

- *例： AsapSource コンポーネントの使用* (317 ページ)

SpPersistentSubscribeSource のプロパティ

SpPersistentSubscribeSource コンポーネントは Event Stream Processor 内のストリームにサブスクライブし、他のコンポーネントにレコードを送信します。

```
ClassName =
com.sybase.esp.adapter.asap.SpPersistentSubscribeSource
```

コンポーネントがサブスクライブするストリームは、サブスクライバによって要求されるまで、レコードを明示的に削除しません。 レコードが処理されると、

SpPersistentSubscribeSource は、タグをプラットフォームにパブリッシュし、サブスクライブされたストリームからローを削除します。

SpPersistentSubscribeSource には、2 つの追加ストリームがあります。 ログ・ストリームとトランケート・ストリームです。たとえば、3 つのストリーム Stream1、Stream1_log、Stream1Truncate を持つことができます。 ログ・ストリームには、2 つの追加カラムがあります。 シーケンス番号と opcode です。 レコードは、Stream1 から Stream1_log に渡され、シーケンス番号値が増加されます。 Stream1_log の opcode カラムの opcode 値は "insert" です。 SpPersistentSubscribeSource がデータのバッチ (または、1 つのレコード) にサブスクライブした後に、レコードの最後のシーケンス番号が Stream1Truncate にパブリッシュされます。その後、そのシーケンス番号より前のレコードが Stream1_log と永続ストア (たとえば、ハード・ディスク上のファイル) から削除されます。

プロパティ	説明
`Host`	(必須) Event Stream Processor の「コマンドと制御」のプロセスのホスト名。
`Port`	(必須) Event Stream Processor の「コマンドと制御」のプロセスのポート番号。
`ProjectUri`	(場合に応じて必須) クラスタ・モードで実行しているサーバに接続する (例：`esp://localost:19011/ws1/p1`)。
`IsEncrypted`	(オプション) このプロパティを true に設定すると、AsapSource は、Event Stream Processor に対して SSL 接続を使用しようとする。
`UseServerRSA`	(オプション) true を指定すると、サーバに接続するためにサーバ RSA 認証が使用される。 このプロパティを指定する場合、`KeyStore` プロパティと `KeyStorePassword` プロパティも指定する必要があります。
`KeyStore`	(オプション) サーバ RSA 認証に使用。 キーストア (.jks ファイル) のロケーションを指定します。 `UseServerRSA` を指定する場合は、このプロパティを設定します。
`KeyStorePassword`	(オプション) キーストアのパスワードを指定。 サーバ RSA 認証に使用されます。 `UseServerRSA` を指定する場合は、このプロパティを設定します。

プロパティ	説明
`UseUserPassword`	(オプション) `User` と `Password` の認証 (たとえば、Kerberos) がサーバに接続するために使用される。 これを指定する場合、`User` と `Password` も指定する必要があります。
`SyncBaseStreams`	(オプション) SYNC_BASE_STREAMS のプラットフォームを問い合わせる。 true に設定すると、各バッチがプラットフォームにパブリッシュされた後に `publisher.commit()` メソッドが呼び出されます。 デフォルト値は true です。
`CommitLimit`	(オプション) 1 回の呼び出しで処理するバッチの最大サイズ。デフォルト値は 100 です。
`User`	(必須) SpPersistentSubscribeSource と Event Stream Processor との間の最初の接続には、認証が必要。 Event Stream Processor に既知の有効なユーザ名を入力します。
`Passwd`	(必須) SpPersistentSubscribeSource と Event Stream Processor との間の最初の接続には、認証が必要。 上記で構成したユーザ名に対する有効なパスワードを入力します。
`Stream++`	(オプション) サブスクライブ先のストリームの名前。
`<Stream++>.TruncateStream`	(必須) データをトランケートするストリーム。
`<Stream++>.OpcodeColumn`	(必須) opcode 値が格納されるカラムの名前。
`<Stream++>.SequenceColumn`	(必須) シーケンス番号が格納されるカラムの名前。

プロパティ	説明
`QueueCapacity`	(オプション) SpPersistentSubscribeSource は、プラットフォームで使用可能なレコードをキューに入れる。 キューに入れられたレコードは、別のスレッドによって使用されます。 このプロパティでは、内部キューの容量を設定します。 キューがいっぱいの場合、アダプタは、領域が使用可能になるまで待機します。 デフォルト値は 4096 です。 **注意：** IsEncrypted プロパティを true に設定すると、ユーザとパスワードの情報が Event Stream Processor に渡されてから、SSL 接続がセットアップされます。 これらの詳細は、プレーン・テキストで渡されます。

参照：
- *例： SpPersistentSubscribeSource コンポーネントの使用* (323 ページ)

シンク・コンポーネント
オープン・アダプタには、2 つのシンク・コンポーネントがあります。 AsapSink と WSSink です。

AsapSink のプロパティ
AsapSink コンポーネントは、ソースからレコードを取得して、そのレコードを Event Stream Processor に配信します。

あらゆる入力アダプタ設定に、1 つの AsapSink コンポーネントが構成されていることを確認してください。

```
ClassName=com.sybase.esp.adapter.asap.AsapSink
```

プロパティ	説明
`ProjectUri`	(場合に応じて必須) クラスタ・モードで実行しているサーバに接続する (例：`esp://localost:19011/ws1/p1`)。
`User`	(必須) AsapSink と Event Stream Processor との間の最初の接続には、認証が必要。 Event Stream Processor に既知の有効なユーザ名を入力します。

プロパティ	説明
`Passwd`	(必須) AsapSink と Event Stream Processor との間の最初の接続には、認証が必要。 上記で構成したユーザ名に対する有効なパスワードを入力します。 **注意：** `UseSSL` プロパティを true に設定すると、ユーザとパスワードの情報が Event Stream Processor に渡されてから、SSL 接続がセットアップされます。 これらの詳細は、プレーン・テキストで渡されます。
`Stream++`	(必須) データの配信先のストリームの名前。
`IsEncrypted`	(オプション) このプロパティが存在し、true に設定すると、AsapSink は、Event Stream Processor に対して SSL 接続を使用しようとする。
`UseServerRSA`	(オプション) true を指定すると、サーバに接続するためにサーバ RSA 認証が使用される。 このプロパティを指定する場合、`KeyStore` プロパティと `KeyStorePassword` プロパティも指定する必要があります。
`KeyStore`	(オプション) サーバ RSA 認証に使用。 キーストア (.jks ファイル) のロケーションを指定します。 `UseServerRSA` を指定する場合は、このプロパティを設定します。
`KeyStorePassword`	(オプション) キーストアのパスワードを指定。 サーバ RSA 認証に使用されます。 `UseServerRSA` を指定する場合は、このプロパティを設定します。
`UseUserPassword`	(オプション) `User` と `Password` の認証 (たとえば、Kerberos) がサーバに接続するために使用される。 これを指定する場合、`User` と `Password` も指定する必要があります。
`SHINE`	(オプション) update 操作と upsert 操作にシャイン・スルーを使用。 これを true に設定し、フィールド値を null に設定すると、既存のレコードを更新しても、そのフィールド値は影響を受けません。

プロパティ	説明
`PublishMethod`	(オプション) データをプラットフォームにパブリッシュする方法を決定。 RECORD、COLLECTION、ENVELOPE、TRANSACTION があります。
`SyncBaseStreams`	(オプション) SYNC_BASE_STREAMS のプラットフォームを問い合わせる。 true に設定すると、各バッチがプラットフォームにパブリッシュされた後に `publisher.commit()` メソッドが呼び出されます。 デフォルト値は false です。
`DiscardFile`	(オプション) 破棄された SDO の移動先のファイルの名前。
`TruncateDiscardFile`	(オプション) true に設定すると、ファイルがトランケートされる。
`EspOpsColumn`	(オプション) このプロパティを設定すると、受信レコードの ESP_OPS 属性の値が、対応するカラムに書き込まれ、そのレコードは、ESP_OPS 値に関係なく、UPSERT として扱われる。

参照：

- *例： AsapSink コンポーネントの使用* (315 ページ)

WSSink のプロパティ

WSSink コンポーネントは、Web サービスのクライアントを実装したもので、リモート・サービスと通信できます。

WSSink は、`lib/WEB-INF/WSAdapterSource.wsdl` の WSDL 記述子と一致します。

注意： アダプタのインストール・ディレクトリの `lib/WEB-INF/WSAdapterSource.wsdl` にある WSDL 記述子に基づいて、サーバ側の Web サービスを構築します。

Web サービス・クライアントは、単純なオブジェクトである Data Transfer Objects (DTO) をデータ・コンテナとして使用します。 使用されるクラスは次のとおりです。

1. `com.sybase.adapter.soap.DataTransferObject`

```
public class DataTransferObject {
private String name;
private int opcode;
```

```
private Object[] data;
}
```

`data` フィールドの構造が、Web サービスで定義されたものと同じであることを確認してください。 DTO のメタデータを取得できます。

2. `com.sybase.adapter.soap.DTOMetaData`

```
public class DTOMetaData {
private String name;
private DTOAttribute[] attributes;
}
```

これは次のクラスを使用します。
`com.sybase.adapter.soap.DTOAttribute`

```
public class DTOAttribute {
private String name;
private String xsdType;
}
```

DataTransferObject、DTOMetaData、DTOAttribute にはいずれも、getter メソッドと setter メソッドが提供されています。 また、サービスの WSDL 記述子からオブジェクト定義を取得できます。

プロパティ	説明
`URL`	(必須) サーバ Web サービスの URL 文字列。 有効な値の例は、"http://eult121.sybase.com:9085/services/WSAdapterSource" です。
`TypeN`	(必須) ソース・コンポーネントがデータを送信するメッセージ・タイプの名前。また、`TypeN` は、公開される DTO の名前です。
`TypeN.<DOType>`	(必須) リモート・サービスで構成される DTO の名前。
`ManualMapping`	(オプション) true の場合、シンクは、設定ファイルで指定されたマッピング AttName->DtoAttName を使用。 false の場合、シンクは、サービスから DTO 情報を取得し、DTO で定義されるすべての属性名が受信アダプタ・メッセージにも存在すると想定します。
`TypeN.AttName++`	(場合に応じてオプション) アダプタ内でソースによって渡されるフィールドの名前。 DTO 属性の名前でもあります。
`TypeN.<AttName++>.DTOAttName`	(場合に応じてオプション) 選択した DTO タイプに対するリモート Web サービスで定義される属性の名前。

プロパティ	説明
`DiscardedLoggerName`	(オプション) 正常に処理されなかったレコードのログ記録を行うロガーの名前。

参照：

- *例： WSSink コンポーネントの使用* (325 ページ)

パイプ・コンポーネント

オープン・アダプタには、2 つのパイプ・コンポーネントがあります。 BeanShellPipe と JDBCLookupPipe です。

BeanShellPipe のプロパティ

メッセージの修正に使用されるスクリプト化可能なパイプ。

メッセージ全体または各フィールドを個別に Java でスクリプト化し、このコンポーネントを受信フローと送信フローで使用できます。 スクリプト化は、BeanShell で実装されています。 この言語の詳細については、http://www.beanshell.org を参照してください。

`ClassName: com.sybase.esp.adapter.scripting.BeanShellPipe`

プロパティ	説明
`MsgPreProcessor`	(必須) このメッセージの BeanShell スクリプト。 スクリプトがメッセージに適用されてから、フィールドが処理されます。 スクリプトは、メッセージ・オブジェクト・タイプにアクセスできます。 変数名は *message* です。 次に例を示します。 `System.out.println("Message received; SDO array size= " + message.peekDataObjects().length);`
`MsgPostProcessor`	(必須) このメッセージの BeanShell スクリプト。 すべてのフィールドが処理されてから、スクリプトがメッセージに適用されます。 クリプトは、メッセージ・オブジェクト・タイプにアクセスできます。 変数名は message です。 次に例を示します。 `System.out.println("Message sent; SDO array size= " + message.peekDataObjects().length);`

プロパティ	説明
Type++	(必須) ソース・コンポーネントから受信するデータ・オブジェクトのタイプ。 受信フロー (AsapSink コンポーネントを介して Event Stream Processor に流れるフロー) の場合、これは、メッセージで更新される Event Stream Processor 基本ストリームの名前です。 送信フロー (Event Stream Processor ストリームからパブリッシュされたデータから発生するフロー) の場合、これは、データをパブリッシュする Event Stream Processor ストリームの名前です。
.PreProcessor	(オプション) このメッセージ・タイプの BeanShell スクリプト。 スクリプトがメッセージに適用されてから、フィールドが処理されます。 スクリプトは、SimpleDataObject オブジェクト・タイプにアクセスできます。 変数名は sdo です。 Typen は、関連する Type プロパティで定義されたメッセージ・タイプまたはストリームの名前です。 次に例を示します。 `System.out.println("Got data for message type: " + sdo.getType().getName());`
.PostProcessor	(オプション) このメッセージ・タイプの BeanShell スクリプト。 すべてのフィールドが処理されてから、スクリプトがメッセージに適用されます。 スクリプトは、SimpleDataObject オブジェクト・タイプにアクセスできます。 変数名は *sdo* です。 Typen は、関連する Type プロパティで定義されたメッセージ・タイプまたはストリームの名前です。 次に例を示します。 `System.out.println("Sending data for message type: " + sdo.getType().getName());`
.AttName++	(必須) 関連付けられたメッセージ・タイプで指定されるフィールド名。 これは、Bean-Shell スクリプトで必須です。 フィールド名が、受け取ったメッセージ・タイプに存在しない場合、新しいフィールドが作成されます。 Typen は、関連する Type プロパティで定義されたメッセージ・タイプまたはストリームの名前です。

プロパティ	説明
`.Script`	(オプション) このフィールドの BeanShell スクリプト。スクリプトは、SimpleDataObject オブジェクト・タイプにアクセスできます。 変数名は *sdo* です。 `Typen` は、関連する `Type` プロパティで定義されたメッセージ・タイプまたはストリームの名前です。
`AttNamex`	(オプション) 関連する `AttName` プロパティで定義されたフィールド名。次に例を示します。 `if (sdo.isPresent("Amount ") && sdo.getAttributeValue ("Amount")>0 {value="AC";} else {value="CO"}`

参照：

- *例： BeanShellPipe コンポーネントの使用* (318 ページ)

JDBCLookupPipe のプロパティ

JDBCLookupPipe コンポーネントは、起動時にデータベースに問い合わせ、キャッシュされた結果セットを検索テーブルとして使用します。

`ClassName: com.sybase.esp.adapter.jdbc.JDBCLookupPipe`

検索テーブルの各レコードは、一意の検索キーと、追加された属性の配列で構成されます。 検索キーは 1 つまたは複数の属性で構成されます。 ソースからデータ・オブジェクトが到着すると、次の動作が行われます。

- キー属性の値を、検索テーブルのレコードと照合する。
- レコードが一致しない場合、そのデータ・オブジェクトは、変換されずにシンクに渡される。
- 検索テーブルのレコードがキー属性の値と一致する場合、検索テーブルから追加された属性がレコードに追加され、レコードの結果がシンクに渡される。

プロパティ	説明
`JdbcDriver`	(必須) データベースへの接続に使用される JDBC ドライバ。次に例を示します。 `oracle.jdbc.OracleDriver`
`JdbcUrl`	(必須) データベースのロケーション。次に例を示します。 `jdbc:oracle:thin:@myhost.com:1521:mydatabase`

プロパティ	説明
`DBProperty++`	(オプション) データベースへの接続時にパイプが設定するデータベース・プロパティの名前。 たとえば、ユーザ名、パスワード、データベース名などです。
`DBPropertyn.Value`	(場合に応じてオプション) 関連する DBProperty の値。 `DBProperty++` プロパティを設定する場合に、このプロパティを設定します。
`Table`	(必須) 検索が実施されるデータベース・テーブルの名前。
`KeyAttName++`	(必須) 検索キーを構成する属性名。
`KeyDbCol++`	(必須) KeyAttNames に相当するデータベース・カラムの名前。
`ValueAttName++`	(必須) 追加される値で使用される属性の名前。
`ValueDbCol++`	(必須) `ValueAttNames` に相当するデータベース・カラムの名前。
`WhereClause`	(オプション) 検索の `SELECT` クエリの一部である `WHERE` 句。 検索クエリは次の形式を使用します。 `SELECT KeyDbCol1, KeyDbCol2, ... , ValueDbCol1, ValueDbCol2, ... FROM Table WHERE WhereClause`

例
Oracle データベース・テーブルの "MyTable = (SYMBOL, ID, PRICE)" を検索に使用します。 各データ・オブジェクトには、4 つの属性があります。 AttA、AttB、AttC、AttD です。 AttA と AttB は、それぞれ SYMBOL と ID に相当し、検索キーとして使用されます。AttD は PRICE に相当し、ソースから受け取られたデータ・オブジェクトに追加されます。 次に、パイプ構成の例を示します。

```
adapter.LOOKUPPIPE.ClassName=
com.sybase.esp.adapter.jdbc.JdbcLookupPipe
adapter.LOOKUPPIPE.JdbcUrl = jdbc:oracle:thin:@myhost.com:
1521:mydatabase
adapter.LOOKUPPIPE.JdbcDriver = oracle.jdbc.OracleDriver
adapter.LOOKUPPIPE.DBProperty1 = user
adapter.LOOKUPPIPE.DBProperty1.Value = MyUser
adapter.LOOKUPPIPE.DBProperty2 = password
adapter.LOOKUPPIPE.DBProperty2.Value = MyPassword
adapter.LOOKUPPIPE.Table = MyTable
adapter.LOOKUPPIPE.KeyDbCol1 = SYMBOL
adapter.LOOKUPPIPE.KeyAttName1 = AttA
adapter.LOOKUPPIPE.KeyDbCol2 = ID
adapter.LOOKUPPIPE.KeyAttName2 = AttB
```

```
adapter.LOOKUPPIPE.ValueDbCol1 = PRICE
adapter.LOOKUPPIPE.ValueAttName1 = AttD
adapter.LOOKUPPIPE.WhereClause = SYMBOL LIKE 'A%'
```

参照：

- *例： JDBCLookupPipe コンポーネントの使用* (319 ページ)

リーダ・コンポーネント

オープン・アダプタには、4 つのリーダ・コンポーネントがあります。MultiFlatXmlStringReader、XPathXmlStreamReader、 XPathMultiTypeXmlReader、EspDelimitedStringReader です。

MultiFlatXmlStringReader のプロパティ

MultiFlatXmlStringReader コンポーネントは、メッセージをすばやく処理し、メッセージの内容に基づいてデフォルトを柔軟に設定でき、データを Event Stream Processor の複数のテーブルに分割します。

このリーダは、単純な XML フォーマットを使用します。テーブル名はタグで、フィールドは属性です。

解析方法として MultiFlatXmlStringReader を選択すると、ソースによって複数のテーブル (定義されたストリーム) に値が設定されます。 Event Stream Processor がソースからのデータに基づいて更新する 1 つまたは複数の内部テーブルを指定します。 また、各フィールドの名前とデータ型を指定して、各ソース・レコードのフィールドを定義します。

MultiFlatXmlStringReader のソース・レコードの形式は次のとおりです。

```
<TableName field1='field1 data' field2='field2 data' … />
```

プロパティ	説明
`AcceptAmpersand`	(デフォルトが必須) true または false の値を入力し、アダプタがアンパサンド (&) の非 XML 使用を受け入れるかどうかを示す。true は、アダプタがアンパサンド (&) の非 XML 使用を受け入れることを示します。 たとえば、アダプタは "&"、"<" などを変換しますが、"Marks & Spencer" などの値も受け入れます。false は、アダプタがアンパサンド (&) の非 XML 使用を拒否することを示します。

プロパティ	説明
`Type++`	(必須) Event Stream Processor がソースのデータに基づいて更新する基本ストリームまたはストリームの名前を入力。更新するストリームごとにこの処理を繰り返します。 各システム・テーブルに `Typen` プロパティがあることを確認してください。
`Typen.AttName++`	(必須) Event Stream Processor がソースのデータに基づいて更新するテーブル・フィールドの名前を入力。 このフィールドは、大文字と小文字を区別します。`Typen` は関連テーブルの名前です。 **注意：** ソース・データの各レコード・フィールドの名前を指定します。
`Typen.AttType++`	(デフォルトが必須) システムで使用される、フィールドのデフォルトのデータ型。`Typen` は関連テーブルの名前です。 Event Stream Processor は、次のデータ型をサポートします。 • `string` – 文字列 • `datetime` – 日付 • `float`– 浮動小数点数 • `short` – 符号付き 16 ビット整数 • `integer` – 符号付き 32 ビット整数 • `long` – 符号付き 64 ビット整数 **注意：** ソース・データの各レコード・フィールドのデータ型を指定します。
`Typen.Format++`	(場合に応じてオプション) 日付時刻データ型のフィールドを作成した場合、そのフィールドのデータを読み込んだときにアダプタが理解するフォーマットを入力。 アダプタは、このフォーマットではないデータを拒否します。`Typen` は関連テーブルの名前です。 値を指定しない場合、アダプタは、そのフィールドに対して、フォーマット `yyyyMMdd` または `yyyyMMdd HH:mm:ss` を持つ日付時刻値のみを理解します。 他のすべての日付時刻データを拒否します。

プロパティ	説明
`Typen.Match`	(必須) このテーブルのレコードに一致する正規表現を入力。`Typen` は関連テーブルの名前です。 各テーブルに対する正規表現を入力します。 次に例を示します。 `.*table_is_x.*`
`Typen.UTCTimeZone++`	(場合に応じてオプション) 日付時刻データ型のフィールドを作成した場合、そのフィールドのタイム・ゾーンを入力可能。`Typen` は関連テーブルの名前です。 アダプタは、対応する日付時刻値を、元のタイム・ゾーン値から、それと等しい UTC 値に変換して正規化します。 次に、この UTC 値が Event Stream Processor に渡され、保存されます。 Java が認識する任意のタイム・ゾーン (たとえば、Europe/London または America/New_York) を入力できます。 値を指定しない場合、日付時刻値は、ローカル時間として Event Stream Processor に渡され、保存されます。

参照：

- *例： MultiFlatXmlStringReader コンポーネントの使用* (321 ページ)
- *オープン・アダプタで有効なタイム・ゾーン* (294 ページ)

XPathXmlStreamReader のプロパティ

XPathXmlStreamReader コンポーネントは、XPath のプロパティを使用して XML 文書を処理します。 解析方法として XPathXmlStreamReader を選択すると、ソースによって複数のテーブルに値が設定されます。

```
DOStringReader=com.sybase.esp.adapter.xml.xpath.XPathXmlStreamReader
```

Event Stream Processor がソースからのデータに基づいて更新する基本ストリームを指定します。 また、各フィールドの名前とデータ型を指定して、各ソース・レコードのフィールドを定義します。

タグ・データまたは属性値を使用して、フィールドに XML 文書からのデータを設定できます。 ネストされたタグを指定するには、スラッシュ (/) を使用してタグ名を区切ります。

たとえば、フィールド・データを xyz に設定します。

```
XPath=/env/body/tag
<env>
<body>
<tag> xyz </tag>
```

```
</body>
</env>
```

属性は、[@attributeName] で指定されます。

この場合、フィールド・データは abc に設定されます。

```
XPath=/env/body/tag[@attr]
<env>
<body>
<tag attr= abc />
</body>
</env>
```

XPathXmlStreamReader はコレクションを処理します。

```
XPath=/env/body/tag
<env>
<body>
<tag> xyz </tag>
<tag> abc </tag>
</body>
</env>
```

デフォルトでは、上記のコマンドは、コレクション・セパレータ文字で区切られた両方の値 (xyz|abc) をフィールドに挿入します。

特定のタグ値が必要な場合、インデックス演算子を使用して、コレクション内の位置を指定します。

```
XPath=/env/body/tag[2]
<env>
<body>
<tag> xyz </tag>
<tag> abc </tag>
</body> </env>
```

このコマンドは、2 番目のタグのみの値を挿入します。

プロパティ	説明
`XmlRoot`	(必須) XML 文書のルート・ノードを入力。次に例を示します。 `env`

プロパティ	説明
`DateFormat`	(オプション) 日付時刻データ型のフィールドを作成する場合、そのフィールドのデータを読み込むときにアダプタが理解するフォーマットを入力。 フィールドの `Format` プロパティで上書きされないかぎり、アダプタは、このフォーマットではないデータを拒否します。 値を入力しない場合、アダプタは、そのフィールドに対して、フォーマット `yyyyMMdd` または `yyyyMMdd HH:mm:ss` を持つ日付時刻値のみを理解します。 他のすべての日付時刻データを拒否します。
`Type++`	(必須) Event Stream Processor がソースのデータに基づいて更新する基本ストリームの名前。
`XPath`	(必須) テーブルのルート・ノードの XPath 形式の表現を入力。次に例を示します。 `/env`
`.AttName++`	(必須) Event Stream Processor がソースのデータに基づいて更新するテーブル・フィールドの名前。 このプロパティは、大文字と小文字を区別します。 **注意：** ターゲットの Event Stream Processor 基本ストリームの各レコード・フィールドの名前を指定します。

`.XPath`	(必須) このフィールドに挿入されるデータの XPath 形式の表現を入力。 この表現が "/" から始まる場合、フル・パスと見なされます。 それ以外の場合は、`Typen.XPath` プロパティに対する相対パスとなります。 次に例を示します。
	`tag`
	または
	`/env/body/tag`
	注意： フル・パスを指定する場合、`Typen.XPath` プロパティより高いレベルでフィールドにアクセスできません。

`.DefaultValue`	(オプション) フィールドが空の場合 (たとえば、空のタグの場合またはタグが XML 文書にない場合)、この値に置換。

プロパティ	説明
`.Format`	(場合に応じてオプション) 日付時刻データ型のフィールドを作成した場合、そのフィールドのデータを読み込んだときにアダプタが理解するフォーマットを入力可能。 アダプタは、このフォーマットではないデータを拒否します。 値を入力しない場合、アダプタは、そのフィールドに対して、フォーマット `yyyyMMdd`、`yyyyMMdd HH:mm:ss`、または `DateFormat` プロパティのデフォルト値の日付時刻値のみを理解します。 他のすべての日付時刻データを拒否します。
`.Match`	(オプション) 必要に応じて、アダプタがレコードに対して照合を実施する正規表現を入力。 AttNamex は、`AttName` プロパティで定義されるフィールド名です。 フィールド・データで正規表現が一致する場合、AttNamex.MatchReplace で定義された文字列に置換されます。次に例を示します。 `.*char_is_(.).*`
`.MatchReplace`	(場合に応じてオプション) 対応する正規表現の照合が成功したときにアダプタが使用できる、`.Match` プロパティに対する置換値を入力。
`.AttType`	(デフォルトが必須) フィールドのデータ型を入力。 `AttNamex` は、`AttName` プロパティで定義されるフィールド名です。 Event Stream Processor は、次のデータ型をサポートします。 • `string` – 文字列 • `datetime` – 日付 • `float`– 浮動小数点数 • `short` – 符号付き 16 ビット整数 • `integer` – 符号付き 32 ビット整数 • `long` – 符号付き 64 ビット整数

プロパティ	説明
`.UTCTimeZone`	(場合に応じてオプション) 日付時刻データ型のフィールドを作成した場合、そのフィールドのタイム・ゾーンを入力可能。 アダプタは、対応する日付時刻値を、元のタイム・ゾーン値から、それと等しい UTC 値に変換して正規化します。 次に、この UTC 値が Event Stream Processor に渡され、保存されます。 Java が認識する任意のタイム・ゾーン (たとえば、Europe/London または America/New_York) を入力できます。 値を指定しない場合、日付時刻値は、ローカル時間として Event Stream Processor に渡され、保存されます。
`BadRecordLoggerName`	(オプション) 不正レコードを書き出すロガーの名前。 動作は実装によって異なります。 名前を指定する場合、実装クラスも指定します。 このプロパティをブランクのままにすると、アダプタは、不正レコードについて警告を表示するだけで、元のデータは失われます。 `<BadRecordLoggerName>.ClassName - Logger implementation`

参照：

- *例： XPathXmlStreamReader コンポーネントの使用* (329 ページ)
- *オープン・アダプタで有効なタイム・ゾーン* (294 ページ)

XPathMultiTypeXmlReader のプロパティ

XPathMultiTypeXmlReader コンポーネントは、XML メッセージを処理します。

```
DOStringReader=com.sybase.esp.adapter.xml.xpath.XPathMultiTypeXmlReader
```

このリーダは、XML で提供されるメッセージ・タイプに応じて、XPathXmlStreamReader を使用します。 メッセージ・タイプを取得すると、このコンポーネントは、標準の XPathXmlStreamReader コンポーネントを使用して、受信メッセージを処理します。 XPathXmlStreamReader コンポーネントのすべての設定プロパティ・ファイルは、解析規則と呼ばれる個別のファイルに保存されます。 解析規則のプロパティのリストは、プレフィクス解析規則が必要なことを除いて、XPathXmlStreamReader に類似しています。

プロパティ	説明
MSGTypeXPath	(必須) XPath として表される XML メッセージのメッセージ・タイプのロケーション。 これは、要素の属性としても提供できます。
MSGTypeN	(必須) メッセージ・タイプの名前。 メッセージ・タイプ名のうちの 1 つは、MSGTypeXPath によって XML メッセージから取得される値に一致する必要があります。
MSGTypeN.ParsingRules	(必須) XPathXmlStreamReader がプロパティを保存するファイルの名前。
BadRecordLoggerName	(オプション) 不正レコードを書き出すロガーの名前。 動作は実装によって異なります。 名前を指定する場合、実装クラスも指定します。 このプロパティをブランクのままにすると、アダプタは、不正レコードについて警告を表示するだけで、元のデータは失われます。 `<BadRecordLoggerName>.ClassName - Logger implementation.`

参照：

- *例： XPathMultiTypeXmlReader コンポーネントの使用* (327 ページ)
- *オープン・アダプタで有効なタイム・ゾーン* (294 ページ)

EspDelimitedStringReader

EspDelimitedStringReader コンポーネントは、区切られた (たとえば、カンマで区切られた) メッセージを処理します。 これを使用して、正しくないレコードを不正レコード・ファイルに送信できます。

```
DOStringReader=com.sybase.esp.adapter.dostrings.EspDelimitedStringReader
```

プロパティ	説明
BadRecordLoggerName	(オプション) 不正レコードを書き出すロガーの名前。 動作は実装によって異なります。 名前を指定する場合、実装クラスも指定します。 このプロパティをブランクのままにすると、アダプタは、不正レコードについて警告を表示するだけで、元のデータは失われます。 `<BadRecordLoggerName>.ClassName - Logger implementation.`

プロパティ	説明

`EmptyStringAsNull`	(オプション) true に設定すると、空の文字列値を null 値に変換。例を示します。
	`NULL`
	入力レコードが次の情報で構成されているとします。
	`A,B,NULL,D`
	その結果得られるフィールド値は次のとおりです。
	`A, B,{null},D`

`FieldDelimiter`	(デフォルトが必須) 1 文字のフィールド・デリミタの文字番号。 16 進数の Unicode 値を表す "uXXXX" または 10 進数の ASCII 値を表す "DDD" を使用します。 このプロパティは、フィールドが非表示文字または空白文字 (たとえば、"スペース"、"タブ"、または "null") の場合に使用します。 デフォルトのデリミタはカンマ (ASCII 44) です。
`NullString`	(オプション) 文字列。フィールド値として使用されると、アダプタによって、生成されるメッセージのこのフィールドに null 文字列が挿入されます。
`StripQuotes`	(オプション) true に設定すると、フィールドが引用符で囲まれている場合、引用符文字がフィールド値の先頭と末尾から削除される。 デフォルト値は true です。

参照：
- *オープン・アダプタで有効なタイム・ゾーン* (294 ページ)

ライタ・コンポーネント
オープン・アダプタには、1 つのライタ・コンポーネント XPathXmlWriter があります。

XPathXmlStringWriter のプロパティ
XPathXmlWriter コンポーネントは、XPath 形式の構文を使用して、パブリッシュされた Event Stream Processor ストリーム・データからの XML 文書をフォーマットします。

フォーマッタは、XML タグと属性をフォーマットします。 このライタを使用するには、シンクが次のプロパティを指定することを確認します。

```
DOStringWriter = com.sybase.esp.adapter.xml.xpath.XmlStringWriter
```

ネストされたタグを指定するには、タグ名を / で区切ります。

```
/env/body/tag
<env>
<body>
<tag>xyz</tag>
</body>
</env>
```

属性は、[@attributeName] で指定されます。

```
/env/body/tag[@attr]
<env>
<body>
<tag attr='xyz'/>
</body>
</env>
```

デフォルトでは、フォーマッタはコレクションを作成します。 たとえば、新しいネストされたタグは、タグ名が出現するたびに作成されます。

```
XPath1=/env/body/tag
XPath2=/env/body/tag
<env>
<body>
<tag>xyz</tag>
</body>
<body>
<tag>abc</tag>
</body>
</env>
```

タグがネスト内に追加される場合、インデックス演算子を使用して、コレクション内の位置を指定します。

```
XPath1=/env/body[1]/tag
XPath2=/env/body[1]/tag
<env>
<body>
<tag>xyz</tag>
<tag>abc</tag>
</body>
</env>
```

このコンポーネントをエンコードする XML コンテンツは iso-8859-1 です。

プロパティ	説明
`Type++`	(必須) エクスポートされる Event Stream Processor ストリームの名前。

`.XPath`	(必須) このテーブルの最上位レベル・タグの XPath 形式の記述。
	`/env/body`
	`<env/>`
	`<body/>`

`.AttName++`	(必須) `Type` (ストリーム) 内のフィールドの名前。

プロパティ	説明

.XPath	(必須) フォーマットされる XML タグまたは属性の XPath 形式の記述。Typen はテーブル名、AttNamen はタグまたは属性に挿入されるデータのソースを指定するフィールド名です。
	記述でタグを指定すると、そのフィールドがタグ・データとして出力されます。
	`/env/body/tag`
	`<env>`
	`<body>`
	`<tag>field data from Typen.AttNamex</tag>`
	`</body>`
	`</env>`
	記述で属性を指定すると、属性値がフィールド・データに設定されます。
	`/env/body/tag[@attr]`
	`<env>`
	`<body>`
	`<tag attr= field data from Typen.AttNamex />`
	`</body>`
	`</env>`

参照：

- *例： XPathXmlStringWriter コンポーネントの使用* (330 ページ)

日付時刻フォーマットの指定

ファイルからデータを読み込むシステムを使用している場合、ファイルに含まれる受け入れ可能な日付のフォーマットを指定できます。

オープン・アダプタは、指定のフォーマットでない日付を拒否します。 受け入れ可能な日付のフォーマットを指定しない場合、アダプタは、フォーマット yyyyMMdd または yyyyMMdd HH:mm:ss を持つ日付時刻値のみを理解し、他の日付時刻データを拒否します。

フォーマットをテンプレート文字列の形で指定する場合、フォーマット文字列と共に、日、年、月などを表す特殊な識別子を指定します。 確実に 24 時間表記を使用するには、時間に大文字の H を使用します。

表 5 : 日付時刻フォーマットの識別子

文字	説明	一般的な使用法
y	年を表す 1 桁の数字	yyyy
M	月を表す 1 桁の数字	MM

文字	説明	一般的な使用法
d	日を表す 1 桁の数字	dd
H	時間 (0 ～ 23) を表す 1 桁の数字	HH
m	分を表す 1 桁の数字	mm
s	秒を表す 1 桁の数字	ss
S	ミリ秒を表す 1 桁の数字	SS

入力データで文字を指定する場合、その文字を一重引用符で囲んで入力してください。 たとえば、入力内容が `Day:2003-12-29 Time:10:22-00` のような文字列の場合、日付時刻フォーマット `'Day' :yyyy-MM-dd 'Time' :HH:mm:ss` を指定します。 この方法でフォーマットを入力することによって、オープン・アダプタが文字をフォーマット指定と間違えなくなります。

日付時刻フォーマットを指定する例を次に示します。

- ファイルから `2003/06/29` (年は 2003、月は 06 (6 月)、日は 29) とフォーマットされたデータを読み込む場合、その日付時刻フォーマットを `yyyy/MM/dd` と入力する。
- ファイルから `29-06-2003 19:12:45` (日は 29、月は 06 (6 月)、年は 2003、時刻は午後 7:12:45) とフォーマットされたデータを読み込む場合、その日付時刻フォーマットを `dd-MM-yyyy HH:mm:ss` と入力する。
- MQ-Series から `2003-06-29 19:12:45.493` (年は 2003、月は 06 (6 月)、日は 29、時刻は午後 7:12:45、493 ミリ秒) のフォーマットの MQPutDateTime の値が渡される場合、その日付時刻フォーマットを `yyyy-MM-dd HH:mm:ss.SSS` と入力する。

注意： オープン・アダプタは、日付時刻から読み込んだミリ秒を削除 (無視) します。

参照：
- *オープン・アダプタで有効なタイム・ゾーン* (294 ページ)

サードパーティ製 JAR ファイル

オープン・アダプタの配布内容には、サードパーティ製の配布可能な JAR ファイルがいくつかあります。

注意： MSSQL JDBC ドライバは配布内容にありません。MSSQL JDBC ドライバは、http://www.microsoft.com/downloads/ja-jp/default.aspx からダウンロードできま

す。「mssql jdbc driver」で検索してください。 オープン・アダプタは、SQL Server 2000 driver for JDBC SP3 をサポートします。

ファイル	説明	ライセンス情報
`.../lib/esp_open_adapter.jar`	Sybase オープン・アダプタ・コンポーネントがある	Sybase
`.../lib/openadaptor.jar`	バージョン 1.7 ベースのオープン・アダプタ・ソースの Sybase 版	https://www.openadaptor.org/licence.html
`.../jar/esp_java_sdk-0.4.jar`	Sybase Event Stream Processor Java SDK ライブラリ	Sybase

`.../lib//jetty/jetty-6.0.1.jar`
`.../lib/jetty/jetty-util-6.0.1.jar`
`.../lib/jetty/servlet-api-2.5-6.0.1.jar`
	Jetty クライアントライブラリ	http://www.apache.org/licenses

`.../lib/bsh-2.0b4.jar`	BeanShell スクリプトの実装があるライブラリ	LGPL http://www.beanshell.org/license.html
`.../jar/commons-codec-1.3.jar`	Apache Commons プロジェクトの一部	Apache ライセンス http://jakarta.apache.org/commons/license.html
`.../jar/commons-discovery-0.2.jar`	Apache Commons プロジェクトの一部	Apache ライセンス http://jakarta.apache.org/commons/license.html
`.../jar/commons-logging-1.1.jar`	Apache Commons プロジェクトの一部	Apache ライセンス http://jakarta.apache.org/commons/license.html

ファイル	説明	ライセンス情報
.../lib/dom4j-1.5.jar	DOM XML 実装	BSD 形式 http://www.dom4j.org/license.html
.../lib/jakarta-oro-2.0.8.jar	Perl5 互換の正規表現、AWK に似た正規表現、glob 表現、ファイル名の置換、分割、フィルタリングを行うユーティリティ・クラスなどを提供する Java クラス。	Apache ライセンス http://svn.apache.org/repos/asf/jakarta/oro/trunk/LICENSE
.../jar/log4j-1.2.14.jar	Java のログ記録実装	Apache ライセンス http://logging.apache.org/log4j/docs/index.html
.../jar/xercesImpl.jar	Xerces2 XML 実装	Apache ライセンス http://xerces.apache.org/xerces-j/
.../jar/xmlrpc-client-3.1.3.jar .../jar/xmlrpc-common-3.1.3.jar .../jar/xmlrpc-server-3.1.3.jar	Java の XML RPC 実装	Apache ライセンス http://ws.apache.org/xmlrpc/
.../jar/adapterapi.jar	外部 Java アダプタ・フレームワーク API	Sybase
.../jar/activation.jar	JavaBeans Activation Framework の仕様	Sun Microsystems
.../jar/axis.jar	SOAP に関する W3C のサブミッションの実装	Apache

ファイル	説明	ライセンス情報
.../jar/esp_java_i18n-1.0.jar	メッセージの国際化	Sybase
.../jar/esp_java_license-1.0.jar	Event Stream Processor のライセンス関連 API	Sybase
.../jar/mail.jar	Java メール API	Sun Microsystems
.../jar/saaj.jar	Web サービス API	Sun Microsystems
.../jar/sylapi.jar	Sybase Event Stream Processor のライセンスに必要	Sybase
.../jar/ws-commons-util-1.0.2.jar	Apache Web サービス共通ユーティリティ	Apache
.../jar/wsdl4j-1.5.1.jar	Java の Web サービスの記述	Apache

オープン・アダプタで有効なタイム・ゾーン
各種アダプタ・リーダ・コンポーネントの `UTCTimeZone` プロパティで考えられる有効なタイム・ゾーンの例。

`UTCTimeZone` プロパティで、Java が認識する任意のタイム・ゾーンを設定できます。 現在、500 を超えるタイム・ゾーンがあります。 Java での包括的なリストを取得するには、TimeZone オブジェクトの `getAvailableIDs()` メソッドを使用します。

```
TimeZone.getAvailableIds()
```

参照：
- *日付時刻フォーマットの指定* (290 ページ)
- *MultiFlatXmlStringReader のプロパティ* (280 ページ)
- *XPathXmlStreamReader のプロパティ* (282 ページ)
- *XPathMultiTypeXmlReader のプロパティ* (286 ページ)
- *EspDelimitedStringReader* (287 ページ)

アフリカのタイム・ゾーン
`UTCTimeZone` プロパティでアフリカに指定する有効なタイム・ゾーン。

国	タイム・ゾーン
アルジェリア	Africa/Algiers
アンゴラ	Africa/Luanda
ベナン	Africa/Porto-Novo
ボツワナ	Africa/Gaborone
ブルキナファソ	Africa/Ouagadougou
ブルンジ	Africa/Bujumbura
カメルーン	Africa/Douala
カーボヴェルデ	Atlantic/Cape_Verde
中央アフリカ共和国	Africa/Bangui
チャド	Africa/Ndjamena
コモロ	Indian/Comoro
コンゴ民主共和国	Africa/Kinshasa Africa/Lubumbashi
コンゴ共和国	Africa/Brazzaville
コートジボワール	Africa/Abidjan
ジブチ	Africa/Djibouti
エジプト	Africa/Cairo
赤道ギニア	Africa/Malabo
エリトリア	Africa/Asmera
エチオピア	Africa/Addis_Ababa
ガボン	Africa/Libreville
ガンビア	Africa/Banjul
ガーナ	Africa/Accra
ギニア	Africa/Conakry

国	タイム・ゾーン
ギニアビサウ	Africa/Bissau
ケニア	Africa/Nairobi
レソト	Africa/Maseru
リベリア	Africa/Monrovia
リビア	Africa/Tripoli
マダガスカル	Indian/Antananarivo
マラウイ	Africa/Blantyre
マリ	Africa/Bamako Africa/Timbuktu
モーリタニア	Africa/Nouakchott
モーリシャス	Indian/Mauritius
マヨット	Indian/Mayotte
モロッコ	Africa/Casablanca
西サハラ	Africa/El_Aaiun
モザンビーク	Africa/Maputo
ナミビア	Africa/Windhoek
ニジェール	Africa/Niamey
ナイジェリア	Africa/Lagos
レユニオン	Indian/Reunion
ルワンダ	Africa/Kigali
セントヘレナ	Atlantic/St_Helena
サントメ・プリンシペ	Africa/Sao_Tome
セネガル	Africa/Dakar
セーシェル	Indian/Mahe
シエラレオネ	Africa/Freetown
ソマリア	Africa/Mogadishu
南アフリカ	Africa/Johannesburg

国	タイム・ゾーン
スーダン	Africa/Khartoum
スワジランド	Africa/Mbabane
タンザニア	Africa/Dar_es_Salaam
トーゴ	Africa/Lome
チュニジア	Africa/Tunis
ウガンダ	Africa/Kampala
ザンビア	Africa/Lusaka
ジンバブエ	Africa/Harare

アジアのタイム・ゾーン
`UTCTimeZone` プロパティでアジアに指定する有効なタイム・ゾーン。

国	タイム・ゾーン
アフガニスタン	Asia/Kabul
アルメニア	Asia/Yerevan
アゼルバイジャン	Asia/Baku
バーレーン	Asia/Bahrain
バングラデシュ	Asia/Dacca
ブータン	Asia/Thimbu
英領インド洋地域	Indian/Chagos
ブルネイ	Asia/Brunei
ビルマ／ミャンマー	Asia/Rangoon
カンボジア	Asia/Phnom_Penh
中国	Asia/Harbin Asia/Shanghai Asia/Chungking Asia/Urumqi Asia/Kashgar

国	タイム・ゾーン
香港	Asia/Hong_Kong
台湾	Asia/Taipei
マカオ	Asia/Macao
キプロス	Asia/Nicosia
グルジア	Asia/Tbilisi
インド	Asia/Calcutta
インドネシア	Asia/Jakarta Asia/Ujung_Pandang Asia/Jayapura
イラン	Asia/Tehran
イラク	Asia/Baghdad
イスラエル	Asia/Jerusalem
日本	Asia/Tokyo
ヨルダン	Asia/Amman
カザフスタン	Asia/Almaty Asia/Aqtobe Asia/Aqtau
キルギス	Asia/Bishkek
韓国と北朝鮮	Asia/Seoul Asia/Pyongyang
クウェート	Asia/Kuwait Asia/Vientiane
レバノン	Asia/Beirut
マレーシア	Asia/Kuala_Lumpur Asia/Kuching
モルディブ	Indian/Maldives

国	タイム・ゾーン
モンゴル	Asia/Dariv Asia/Ulan_Bator Asia/Baruun-Urt
ネパール	Asia/Katmandu
オマーン	Asia/Muscat
パキスタン	Asia/Karachi
パレスチナ	Asia/Gaza
フィリピン	Asia/Manila
カタール	Asia/Qatar
サウジアラビア	Asia/Riyadh
シンガポール	Asia/Singapore
スリランカ	Asia/Colombo
シリア	Asia/Damascus
タジキスタン	Asia/Dushanbe
タイ	Asia/Bangkok
トルクメニスタン	Asia/Ashkhabad
アラブ首長国連邦	Asia/Dubai
ウズベキスタン	Asia/Samarkand Asia/Tashkent
ベトナム	Asia/Saigon
イエメン	Asia/Aden

大洋州のタイム・ゾーン
`UTCTimeZone` プロパティで大洋州に指定する有効なタイム・ゾーン。

国	タイム・ゾーン
オーストラリア	Australia/Adelaide Australia/Brisbane Australia/Broken_Hill Australia/Darwin Australia/Hobart Australia/Lindeman Australia/Lord_Howe Australia/Melbourne Australia/Perth Australia/Sydney
クリスマス島	Indian/Christmas
クック諸島	Pacific/Rarotonga
ココス諸島	Indian/Cocos
フィジー	Pacific/Fiji
仏領ポリネシア	Pacific/Gambier Pacific/Marquesas Pacific/Tahiti
グアム	Pacific/Guam
キリバス	Pacific/Tarawa Pacific/Enderbury Pacific/Kiritimati
北マリアナ諸島	Pacific/Saipan
マーシャル諸島	Pacific/Majuro Pacific/Kwajalein

国	タイム・ゾーン
ミクロネシア	Pacific/Yap Pacific/Truk Pacific/Ponape Pacific/Kosrae
ナウル	Pacific/Nauru
ニューカレドニア	Pacific/Noumea
ニュージーランド	Pacific/Auckland Pacific/Chatham
ニウエ	Pacific/Niue
ノーフォーク	Pacific/Norfolk
パラオ (ベラウ)	Pacific/Palau
パプアニューギニア	Pacific/Port_Moresby
ピトケアン島	Pacific/Pitcairn
米領サモア	American Samoa
サモア	Pacific/Apia
ソロモン諸島	Pacific/Guadalcanal
トケラウ諸島	Pacific/Fakaofo
トンガ	Pacific/Tongatapu
ツバル	Pacific/Funafuti
米領太平洋諸島	Pacific/Johnston Pacific/Midway Pacific/Wake
バヌアツ	Pacific/Efate
ワリス・フテュナ	Pacific/Wallis

欧州のタイム・ゾーン
`UTCTimeZone` プロパティで欧州に指定する有効なタイム・ゾーン。

国	タイム・ゾーン
アンドラ	Europe/Andorra
オーストリア	Europe/Vienna
ベラルーシ	Europe/Minsk
ベルギー	Europe/Brussels
英国／アイルランド	Europe/London Europe/Belfast Europe/Dublin
ブルガリア	Europe/Sofia
チェコ共和国	Europe/Prague
デンマーク、フェロー諸島、グリーンランド	Europe/Copenhagen Atlantic/Faeroe
エストニア	Europe/Tallinn
フィンランド	Europe/Helsinki
フランス	Europe/Paris
ドイツ	Europe/Berlin
ジブラルタル	Europe/Gibraltar
ギリシャ	Europe/Athens
ハンガリー	Europe/Budapest
アイスランド	Atlantic/Reykjavik
イタリア	Europe/Rome Europe/San_Marino Europe/Vatican
ラトビア	Europe/Riga
リヒテンシュタイン	Europe/Vaduz
リトアニア	Europe/Vilnius

国	タイム・ゾーン
ルクセンブルク	Europe/Luxembourg
マルタ	Europe/Malta
モルドバ	Europe/Chisinau
モナコ	Europe/Monaco
オランダ	Europe/Amsterdam
ノルウェー	Europe/Oslo
ポーランド	Europe/Warsaw
ポルトガル	Europe/Lisbon Atlantic/Azores Atlantic/Madeira
ルーマニア	Europe/Bucharest
ロシア	Europe/Kaliningrad Europe/Moscow Europe/Samara Asia/Yekaterinburg Asia/Oms Asia/Krasnoyarsk Asia/Irkutsk Asia/Yakutsk Asia/Vladivostok Asia/Magadan Asia/Kamchatka Asia/Anadyr
スペイン	Africa/Ceuta Atlantic/Canary Europe/Madrid
スウェーデン	Europe/Stockholm
スイス	Europe/Zurich

国	タイム・ゾーン
トルコ	Europe/Istanbul Asia/Istanbul
ウクライナ	Europe/Kiev Europe/Simferopol
ユーゴスラビア	Europe/Belgrade Europe/Ljubljana Europe/Sarajevo Europe/Skopje Europe/Zagreb

北米のタイム・ゾーン
`UTCTimeZone` プロパティで北米に指定する有効なタイム・ゾーン。

国	タイム・ゾーン
アンギラ	America/Anguilla
アンティグア・バーブーダ	America/Antigua
バハマ	America/Nassau
バルバドス	America/Barbados
ベリーズ	America/Belize
バミューダ諸島	Atlantic/Bermuda
英領バージン諸島	America/Tortola

国	タイム・ゾーン
カナダ	America/Dawson America/Dawson_Creek America/Edmonton America/Glace_Bay America/Goose_Bay America/Halifax America/Inuvik America/Iqaluit America/Montreal America/Nipigon America/Pangnirtung America/Rainy_River America/Rankin_Inlet America/Regina America/St_Johns America/Swift_Current America/Thunder_Bay America/Vancouver America/Whitehorse America/Winnipeg America/Yellowknife
ケイマン諸島	America/Cayman
コスタリカ	America/Costa_Rica
キューバ	America/Havana
ドミニカ	America/Dominica
ドミニカ共和国	America/Santo_Domingo
エルサルバドル	America/El_Salvador
グレナダ	America/Grenada

国	タイム・ゾーン
グアドループ島	America/Guadeloupe
グアテマラ	America/Guatemala
ハイチ	America/Port-au-Prince
ホンジュラス	America/Tegucigalpa
ジャマイカ	America/Jamaica
マルチニーク島	America/Martinique
メキシコ	America/Cancun America/Chihuahua America/Ensenada America/Mazatlan America/Mexico_City America/Tijuana
モントセラト	America/Montserrat
ニカラグア	America/Managua
パナマ	America/Panama
プエルトリコ	America/Puerto_Rico
セントクリストファー・ネーヴィス	America/St_Kitts
セントルシア	America/St_Lucia
サンピエール島・ミクロン島	America/Miquelon
セントビンセントおよびグレナディーン諸島	America/St_Vincent
タークス諸島とカイコス諸島	America/Grand_Turk
米国	America/Chicago America/Denver America/Honolulu America/Los_Angeles America/New_York

国	タイム・ゾーン
米国 (アラスカ)	America/Adak America/Anchorage America/Juneau America/Nome America/Yakutat
米国 (上記を除く)	America/Boise America/Detroit America/Indiana/Knox America/Indiana/Marengo America/Indiana/Vevay America/Indianapolis America/Louisville America/Menominee America/Phoenix
バージン諸島	America/St_Thomas

南米のタイム・ゾーン
`UTCTimeZone` プロパティで南米に指定する有効なタイム・ゾーン。

国	タイム・ゾーン
アルゼンチン	America/Buenos_Aires America/Catamarca America/Cordoba America/Jujuy America/Mendoza America/Rosario
アルバ	America/Aruba
ボリビア	America/La_Paz

国	タイム・ゾーン
ブラジル	America/Araguaina America/Belem America/Cuiaba America/Fortaleza America/Maceio America/Manaus America/Noronha America/Porto_Acre America/Porto_Velho America/Sao_Paulo
チリ	America/Santiago Pacific/Easter
コロンビア	America/Bogota
キュラソー島	America/Curacao
エクアドル	America/Guayaquil Pacific/Galapagos
フォークランド諸島	Atlantic/Stanley
仏領ギアナ	America/Cayenne
ガイアナ	America/Guyana
パラグアイ	America/Asuncion
ペルー	America/Lima
サウスジョージア島	Atlantic/South_Georgia
スリナム	America/Paramaribo
トリニダード・トバゴ	America/Port_of_Spain
ウルグアイ	America/Montevideo
ベネズエラ	America/Caracas

オープン・アダプタの起動

オープン・アダプタを bootstrap クラスを介して起動します。

前提条件

1. Java Runtime Environment 1.6.0_26 以降をインストールする。
2. JAVA_HOME 環境変数を JRE 1.6.0_26 ルートに設定する。

手順

オープン・アダプタを bootstrap クラスを介して起動します。このクラスは、設定ファイルを読み込み、アダプタ・コンポーネントを起動します。

```
java - Xmx768M –cp ClassPath Bootstrap PropertyFile
AdapterName
```

各パラメータの意味は次のとおりです。

- **`Xmx768M`** – メモリ・ヒープのサイズを指定する Java パラメータ。 多くのメモリを必要とするアダプタ構成では、メモリ・サイズを 768 から増やせます。
- **`ClassPath`** – オープン・アダプタで必要な JAR ファイルまたはクラスが格納されている Java クラスパス。 ここには、サードパーティの JAR ファイルもあります。 使用されるクラスパスの詳細については、`examples` ディレクトリ内のコンポーネント・プロパティ・ファイルを参照してください。
- **`Bootstrap`** – コマンド・ライン・プログラムとして実行できるアダプタ・ブートストラップと Java クラス。 提供されるクラスの名前は、`org.openadapter.adapter.RunAdaptor` です。
- **`PropertyFile`** – アダプタ `<adapterName>.props` のコンポーネント設定で構成されるプロパティ・ファイルの名前。
- **`AdapterName`** – 起動するアダプタの名前。 アダプタ設定を作成する場合、アダプタにわかりやすい名前を付け、アダプタ・プロセスを容易に識別し、モニタできるようにすることをおすすめします。

オープン・アダプタのモニタリング

RemoteControl と RemoteLogger のインタフェースを使用して、実行中のアダプタをモニタします。

RemoteControl と RemoteLogger はオプションのアダプタ実装です。 これらのインタフェースをアダプタ・プロパティで指定します。

実行中のアダプタには、動的な制御インタフェースをサポートし、RemoteControl によって呼び出されるコントローラがあります。RemoteControl には、オペレータが実行中のアダプタと通信を確立し、現在のステータスを確認し、問題を解決するためのインタフェースが用意されています。 RemoteControl を指定するには、次のプロパティ構文を使用します。

```
adapterName. Controller.RemoteControl.ClassName = Class
```

リモート・ロギングは、アダプタのログ出力をフィルタし、警告またはメッセージを生成します。 リモート・ロガーを設定して、FATAL、WARN などの特定のログ・レベルのすべてのログ行を送信できます。 正規表現を使用して、明示的なエラーに対してパターン一致する行を選択できます。 RemoteLogger を指定するには、次の構文を使用します。

```
adapterName. Logging.RemoteLogger.ClassName = Class
```

Sybase では、リモート制御とリモート・ロギングのために、いくつかの標準機能を実装しています。

表 6 : リモート制御とリモート・ロギングのインタフェース実装

インタフェース	説明
HTTPRemote-Control	アダプタを Web サーバにする。 実行中のアダプタが存在するマシンと、HTTPRemoteControl が受信しており、HTTPRemote-Control が簡易な制御インタフェースを返すポート (デフォルト値は 80) を、ブラウザで指定します。
RvRemote-Control	TIBCO Rendezvous を使用して、アダプタが実行している場所が不明でもアダプタと通信できるようにする。
JMXRemote-Control	JMX 準拠のリモート制御。
RMIRemote-Control	RMI サービスとして登録され、クライアントが RMI を使用してアダプタのサポート、管理、設定を実行できるようにする。
RvRemoteLogger	特定の Rendezvous サブジェクトに関するログ行をパブリッシュする。
MailRemote-Logger	Java Mail API を使用して電子メール・メッセージを送信する。

リモート制御インタフェース

リモート制御実装は、要求を、特定の属性で構成される DataObject として受信することを想定します。

表 7 : リモート制御インタフェース属性

属性	説明
Method	オペレーションの名前。
UserName	要求を送信しているユーザの名前。
HostName	要求の送信元ホストの名前。

属性	説明
Comment	任意のコメント。
ControllerName	アダプタ・コントローラの名前。 たとえば、Adapter.Controller です。
Password	設定する場合、アダプタの `ControlPassword` プロパティに一致することを確認する。
Arg1…ArgN	メソッドの引数。

表 8 : リモート制御メソッド (オペレーション)

オペレーション	説明
pause	`pause()` をすべてのアダプタ・コンポーネントで起動。 アダプタが再開を指示されるまで、いずれのメッセージも処理されません。
resume	`resume()` をすべてのアダプタ・コンポーネントで起動。 メッセージ処理が再開されます。
terminate	`terminate()` をすべてのアダプタ・コンポーネントで起動。 すべてのソース・コンポーネントがコントローラに終了処理を行っていることを通知し、その後、コントローラが終了します。
kill	`System.exit(0)` を起動し、コントローラ・プロセスを終了する。
logLines	アダプタの出力ロガーから最後の N 行を返す。 OutputLogger は、書き込んだ最後の N 行をキャッシュに保持します。 デフォルトは 10 です。この値は `LogLinesToCache` プロパティを使用して変更できます。
status	`getStatus()` を起動する。すべてのコンポーネントについての文字列を返し、制御インタフェースの応答サブジェクトについての統合されたステータス・メッセージをパブリッシュします。 その後、制御ユーティリティは結果を表示できます。 独自のコンポーネントを記述して、`getStatus()` メソッドを上書きできます。
setLogLevel	アダプタの OutputLogger で `setLogLevel(arg1,arg2)` を起動する。`setLogLevel` は、要求 DataObject の arg1 と arg2 がログレベルとスコープであると想定します。 例は、INFO DEFAULT です。
customControl	要求 DataObject の arg1 がアダプタ・コンポーネント名であると想定。 リモート制御によって、コンポーネントに対して `customControl()` が呼び出され、要求 DataObject 全体がコンポーネントに転送されます。 この操作を編集できます。

HTTPRemoteControl

HTTPRemoteControl の実装によって HTTP ベースのリモート制御が実現され、アダプタが単純な Web サーバになります。

リモート制御では、定義済みポート (デフォルトは 80) で HTTP 要求を受信し、単純な HTML インタフェースを返します。 このインタフェースは制御パネルで、ボタンを押すと HTTP get 要求が生成されます。 リモート制御は、これらの URL を解析して、リモート制御要求を生成します。

アダプタ A については、以下を設定します。

```
A.Controller.RemoteControl.ClassName =
org...standard.HTTPRemoteControl
A.Controller.RemoteControl.HTTPPort = ?
A.Controller.RemoteControl.ControlPassword = ?
```

`HTTPort` プロパティと `ControlPassword` プロパティはオプションです。 これらのプロパティのデフォルトは、ポート 80 とパスワードなしです。

このリモート制御をテストするには、ブラウザに次の URL を入力します。
`http://<hostname>:<port>`

動的制御インタフェースをサポートする制御パネルが表示されます。 この制御インタフェースは URL 解析に基づいており、URL をカット・アンド・ペーストして、それを既存の Web サイトに対して使用できます。 応答時に制御インタフェースを使用不可能にするには、URL に `&reply=false` を追加します。

URL の構文は次のとおりです。

```
 http:// HostName :Port/ ?name= ControllerName &method= Method
&password= ControlPassword &arg1= Arg-Value ... &argN= ArgValue
&reply={true|false}
```

HTTPRemoteControl は、この URL を解析し、要求 DataObject を生成します。この要求は、AbstractRemoteControl によって処理されます。

MailRemoteLogger

MailRemoteLogger の実装では、Java Mail API を使用します。

CLASSPATH には、`mail.jar` と `activation.jar` を含める必要があります。 アダプタ A については、以下を設定します。

```
A.Logging.RemoteLogSetting = FATAL A.Logging.RemoteLogger.ClassName
=
org.openadapter.adapter.mail.MailRemoteLogger
A.Logging.RemoteLogger.Mailhost = mailhost@foo.com
A.Logging.RemoteLogger.To = fred@openadaptor.org
A.Logging.RemoteLogger.FilterPattern = failed to connect
```

注意： `FilterPattern` はオプションのプロパティですが、正規表現を使用する必要があります。 追加のプロパティについては、Java のマニュアルを参照してください。

標準的なオープン・アダプタ・コントローラには、セキュリティ関連のあらゆる問題を扱う OASecurityManager インタフェースが用意されています。
OASecurityManager の実装を選択するには、コントローラ・プロパティ・ファイルの `SecurityManager.ClassName` コントローラ・プロパティを設定します。

PasswordEncryptor

PasswordEncryptor コンポーネントは、オープン・アダプタ・コンポーネントにプレーン・テキストのパスワードが存在しないようにします。

Event Stream Processor のオープン・アダプタ用拡張機能によって、サンプル・キーストアに、プライベート・キーとパブリック・キーのペアが追加されます。 キーストアのデフォルト・ロケーションは `$ESP_HOME/adapters/esp_open/lib/security` です。 次の 3 つのサンプルがあります。

- `jksKeyStore` – 512 ビットの暗号化強度で生成された RSA キー・ペアがある Java ネイティブ・キーストア。 キーストアのパスワードは "changeit"、キー・ペアのエイリアスは "adaptor" です。
- `pkcs8KeyStore.der` – PKCS#8 標準の形式で、DER を使用して暗号化されたキーストア。 パスワードもエイリアスも想定されていません。 RSA キーペアがあり、512 ビットの暗号化強度を使用して生成されます。
- `pkcs12KeyStore.p12` – PKCS#12 標準の形式のキーストア。 パスワードは "changeit"、エイリアスは "adaptor" です。

注意： 上記のキーストアは単なるサンプルです。 運用システムでは、お客様所有のキーを使用してください。

オープン・アダプタには、パスワード文字列を暗号化するための簡単なツールが用意されています。 パスワードを暗号化するには、`$ESP_HOME/adapters/esp_open/bin` の `pwdenc.sh` ファイルと `pwdenc.bat` ファイルを使用します。 このツールでは、次の 2 つのパラメータが必須です。

- **`-t`** – キーストアのタイプ。 有効な値は、JKS、PKCS8、PKCS12 です。
- **`-k`** – キーストアのロケーション。

何も設定しない場合、次のデフォルト値が使用されます。

```
pwdenc -t JKS -k ../lib/security/jksKeyStore.der
```

キーストアのタイプに応じて、さらに質問が表示されます。 暗号化されたパスワードは、シェル・スクリプトを実行したディレクトリの `encrytedPwd.txt` ファイルに保存されます。 たとえば、`$ESP_HOME/adapters/esp_open/bin`

です。 また、文字列は、base64 アルゴリズムを使用して暗号化されます。 制限事項は、アダプタ・プロパティ・ファイルの 1 行にすべての文字を記述する必要があることです。 暗号化形式のパスワードを、アダプタ・プロパティ・ファイルのコンポーネントの関連パスワード・フィールドにコピーする必要があります。

プロパティ	説明
`KeyStore`	(必須) キーストア・ファイルのロケーション。
`KeyStoreType`	(オプション) キーストア・ファイルの保存に使用される標準。有効な値は次のとおりです。 JKS、PKCS8、PKCS12。デフォルト値は JKS です。
`KeyAlias`	(オプション) キーストア・タイプが JKS または PKCS12 の場合、キー・ペアのエイリアス名を指定。 このプロパティは、PKCS8 では使用されません。
`KeyStorePassword`	(オプション) キーストア・タイプが JKS または PKCS12 の場合、パスワードを指定。 このプロパティは、PKCS8 では使用されません。

Java の keytool を使用した自己署名 RSA キーの生成
Java の keytool を使用して自己署名 RSA キーを生成するには、サンプルである、`$ESP_HOME/adapters/esp_open/lib/security` ディレクトリの `jksKeyStore` ファイルを使用します。

コマンド・プロンプトで次を実行します。

```
keytool -genkey -keyalg rsa -keysize 512 -alias adaptor -keystore
jksKeyStore
```

OpenSSL を使用した自己署名 RSA キーの生成
OpenSSL を使用して自己署名 RSA キーを生成するには、`$ESP_HOME/adapters/esp_open/lib/security` ディレクトリの `PKCS12 Keystore` ファイルを使用します。

1. CA プライベート・キーを生成します。
   ```
   openssl genrsa -rand -des3 -out ca.key 512
   ```
2. そのキーを使用して CA 証明書を作成します。
   ```
   openssl req -new -x509 -days 365 -key ca.key -out ca.pem -outform
   PEM
   ```
3. 作成した CA 証明書を clientTrustStore にインポートできるようにするために、これをエクスポートします。

```
openssl x509 -in ca.pem -out caCert.pem -outform PEM -signkey
ca.key
```

4. サーバのプライベート・キーを生成します。
```
openssl genrsa -rand -des3 -out server.key 512
```
5. サーバ証明書を作成します。
```
openssl req -new -days 365 -key server.key -out server.crs
```
6. 作成した CA 証明書を使用してサーバ証明書に署名します。
```
openssl ca -in server.crs -out signedServerCert.pem -keyfile
ca.key -cert caCert.pem
```
7. この証明書をキュー・マネージャ・ストアにインポートできるようにするために、PKCS#12 フォーマットでエクスポートします。
```
openssl pkcs12 -export -in signedServerCert.pem -out
pkcs12KeyStore.p12 -inkey server.key -name adaptor
```

OpenSSL を使用した自己署名 RSA キーの生成 (PKCS8 キーストア)
OpenSSL を使用して自己署名 RSA キーを生成するには、サンプルである、`$ESP_HOME/adapters/esp_open/lib/security` ディレクトリの `pkcs8KeyStore.der` ファイルを使用します。

コマンド・プロンプトで次のように入力します。
```
openssl pkcs8 -nocrypt -in server.key -out pkcs8KeyStore.der -
outform DER -topk8
```

例
さまざまなオープン・アダプタ・コンポーネントを動作させる方法の例を示します。

例：AsapSink コンポーネントの使用
FilePollSource (リーダ) コンポーネントと AsapSink (ライタ) コンポーネントを関連付けます。 FilePollSource コンポーネントは、ディスク上のファイル (insert.txt) からレコードを読み込み、それらのレコードを AsapSink コンポーネントに転送します。 次に、AsapSink コンポーネントがそれらのレコードを Event Stream Processor にパブリッシュします。

1. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`

Windows	コマンド・ウィンドウを開く。
	1. サンプル・クラスタを起動。
	`start_server_cluster.bat`
	2. プロジェクトをクラスタに追加し、クラスタ上で起動。
	`start_project.bat`

2. **esp_subscriber** を起動して、クラスタで実行している上記のプロジェクトにサブスクライブします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./esp-subscribe.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `esp-subscribe.bat`

3. FilePollsource コンポーネントと AsapSink コンポーネントを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./fileToAsap.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **fileToAsap.bat**

   FilePollSource コンポーネントは、`insert.txt` ファイルから読み込み、レコードを AsapSink コンポーネントに転送して、サーバにパブリッシュします。

4. `fileToAsap.props` ファイルの `adaptor.FILESOURCE.InputFileName` を `insert_withNULL.txt` に変更し、再実行します。

参照：

- *AsapSink のプロパティ* (272 ページ)

例：AsapSource コンポーネントの使用

AsapSource (リーダ) コンポーネントと FileSink (ライタ) コンポーネントを関連付けます。 AsapSource は、Event Stream Processor からレコードを読み込み、それらを FileSink に渡します。FileSink は、それらのレコードを `out.txt` ファイルに書き込みます。

1. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`

Windows	コマンド・ウィンドウを開く。
	1. サンプル・クラスタを起動。
	`start_server_cluster.bat`
	2. プロジェクトをクラスタに追加し、クラスタ上で起動。
	`start_project.bat`

2. **esp_subscriber** を起動して、クラスタで実行している上記のプロジェクトにサブスクライブします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./esp-subscribe.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `esp-subscribe.bat`

3. AsapSource コンポーネントと FileSink コンポーネントを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./asapToFile.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **asapToFile.bat**

4. `esp_insert.txt` ファイルからのデータをサーバにアップロードします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./esp_upload.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **esp_upload.bat**

   AsapSource は、このパブリッシュされたデータをサーバから読み込み、FileSink に渡します。FileSink は、これを `out.txt` ファイルに書き込みます。

参照：

- *AsapSource のプロパティ* (267 ページ)

例：BeanShellPipe コンポーネントの使用

AsapSource コンポーネントと FileSink コンポーネントの間で BeanShellPipe コンポーネントを使用できます。 BeanShellPipe は、AsapSource からデータを受け取ってから、シェルでいくつかのコマンドを実行し、その後で FileSink にデータをパブリッシュします。

1. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。`start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`
Windows	コマンド・ウィンドウを開く。 1. サンプル・クラスタを起動。`start_server_cluster.bat` 2. プロジェクトをクラスタに追加し、クラスタ上で起動。`start_project.bat`

2. **esp_subscriber** を起動して、クラスタで実行している上記のプロジェクトにサブスクライブします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./esp-subscribe.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `esp-subscribe.bat`

3. AsapSource コンポーネント、FileSink コンポーネント、BeanShellPipe コンポーネントを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./asapToFile.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **asapToFile.bat**

4. `esp_insert.txt` ファイルからのデータをサーバにアップロードします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./esp_upload.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **esp_upload.bat**

   AsapSource は、このデータを読み込み、BeanShellPipe に渡します。次に、BeanShellPipe はこれを FileSink に渡し、FileSink はこれを `out.txt` ファイルに書き込みます。BeanShellPipe は、テキストをコマンド・プロンプトに出力します。

参照：

- *BeanShellPipe のプロパティ* (276 ページ)

例：JDBCLookupPipe コンポーネントの使用

AsapSource は、Event Stream Processor からデータを読み込み、それを JDBC 検索パイプに渡します。 JDBC 検索パイプは、必要に応じて、'replaceValue1' を使用して 'charfield' カラムの値を変更し、そのデータを FileSource に渡します。次に、FileSource は、そのデータを `file out.txt` ファイルに出力します。

1. テーブルを作成し、そのテーブルにデータを作成します。 たとえば、DB2 データベースでは、**createTable_DB2.sql** スクリプトを実行します。
   このスクリプトを修正して、他のあらゆるデータベースで使用します。
2. `JdbcLookupPipe.props` ファイルの DB プロパティを、必要なデータベース・インスタンスを指すように更新します。
3. `JdbcLookupPipe.bat` スクリプトまたは `JdbcLookupPipe.sh` スクリプトを修正し、クラスパスに JDBC ドライバの JAR を追加します。
4. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`
Windows	コマンド・ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.bat` 2. プロジェクトをクラスタに追加し、クラスタ上で起動。 `start_project.bat`

5. **esp_subscriber** を起動して、クラスタで実行している上記のプロジェクトにサブスクライブします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./esp-subscribe.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `esp-subscribe.bat`

6. AsapSource コンポーネント、FileSink コンポーネント、JDBCLookupPipe コンポーネントを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./JdbcLookupPipe.sh**

オペレーティング・システム	手順
Windows	コマンド・ウィンドウを開き、次を入力。 **JdbcLookupPipe.bat**

7. サーバにデータをアップロードします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./esp_upload.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **esp_upload.bat**

   AsapSource は、このデータ (レコード) を読み込み、それを JDBCLookupPipe に渡します。JDBCLookupPipe は、それらのレコードを、データベース・テーブルから利用可能なデータと参照データに従って修正します。 JDBCLookupPipe は次にそのデータを FileSink に渡し、FileSink はレコードをファイルに書き込みます。
   - テーブル "test1" には、データ "col1='AttributeKey'" と "col2='replaceValue1'" がある。 'KeyDbCol1' は props ファイルの col1 なので、col1 カラムには属性キーがあります。
   - これらの属性キーは、受信レコード・カラムの 'textfield' にある。
   - レコードの 'charfield' カラムの値を 'replaceValue1' に置換するには、レコードの 'textfield' カラムの値として 'AttributeKey' を構成する。

   詳細については、`esp_insert.txt` ファイルを参照してください。 'textfield' カラム値として 'AttributeKey' を持たないレコードは修正されません。
8. `out.txt` ファイルの内容を参照してください。

   一部のレコードの charfield データが 'replaceValue1' 値に更新されています。

参照：
- *JDBCLookupPipe のプロパティ* (278 ページ)

例： MultiFlatXmlStringReader コンポーネントの使用

MultiFlatXmlStringReader コンポーネントと FilePollSource コンポーネントを関連付けることによって、XML フォーマットのレコードを読み込んで AsapSink に渡し、AsapSink がそれらをサーバにパブリッシュできます。

1. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`

Windows	コマンド・ウィンドウを開く。
	1. サンプル・クラスタを起動。
	`start_server_cluster.bat`
	2. プロジェクトをクラスタに追加し、クラスタ上で起動。
	`start_project.bat`

2. **esp_subscriber** を起動して、クラスタで実行している上記のプロジェクトにサブスクライブします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./esp-subscribe.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `esp-subscribe.bat`

3. MultiFlatXmlStringReader コンポーネント、FilePollSource コンポーネント、AsapSink コンポーネントを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./MultiFlatXMLStringReader.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **MultiFlatXMLStringReader.bat**

FilePollSource は、XML にフォーマットされたデータを `insert.xml` ファイルから MultiFlatXmlStringReader を使用して読み込み、AsapSink に渡します。AsapSink は、これをサーバにパブリッシュします。

参照：

- *MultiFlatXmlStringReader のプロパティ* (280 ページ)

例： SpPersistentSubscribeSource コンポーネントの使用
SpPersistentSubscribeSource コンポーネントは、永続サブスクライブを使用してサーバをサブスクライブします (サブスクライブされたレコードは、処理されるまで保存され、処理後に削除されます)。

これを実装するには、ログ・ストリーム (Stream1_log) とトランケート・ストリーム (TruncateStream1) をストリーム "Stream1" に対して作成します。 Stream1_log はデータを保存し、TruncateStream1 にはプライマリ・キーとシーケンス番号の 2 つのカラムがあります。 詳細については、`bin` フォルダの `model.ccl` を参照してください。

受信レコードは、追加のシーケンス番号カラムと共に Stream1_log に転送されます。 Stream1_log からのレコードが処理されると、最後のシーケンス番号が TruncateStream1 にパブリッシュされます。 次に、このパブリッシュされたシーケンス番号以下のシーケンス番号を持つすべてのレコードが Stream1_log から削除されます。

1. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`
Windows	コマンド・ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.bat` 2. プロジェクトをクラスタに追加し、クラスタ上で起動。 `start_project.bat`

2. **esp_subscriber** を起動して、上記のクラスタで動作しているプロジェクトの Stream1 にサブスクライブします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./esp-subscribe-Stream1.sh**

オペレーティング・システム	手順
Windows	コマンド・ウィンドウを開き、次を入力。 **esp-subscribe-Stream1.bat**

3. ログ・ストリーム Stream1_Log にサブスクライブします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./esp-subscribe-Stream1_log.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **esp-subscribe-Stream1_log.bat**

4. ログ・ストリーム Truncate_stream1 にサブスクライブします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./esp-subscribe-TruncateStream1.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **esp-subscribe-TruncateStream1.bat**

5. SpPersistentSubscribeSource コンポーネントと FileSink コンポーネントを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./asapToFile.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **asapToFile.bat**

6. `esp_insert.txt` ファイルからのデータをサーバにアップロードします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./esp_upload.sh**

オペレーティング・システム	手順
Windows	コマンド・ウィンドウを開き、次を入力。 **esp_upload.bat**

SpPersistentSubscribeSource は、サーバと Stream1_log にサブスクライブし、それらのレコードを FileSink に渡します。FileSink は、それらのレコードを `out.txt` ファイルに書き込みます。 すべてのサブスクリプション・スクリプト・ファイルには、それぞれのサブスクリプションが記述されます。

参照：

- *SpPersistentSubscribeSource のプロパティ* (269 ページ)

例：WSSink コンポーネントの使用

`WsSink.props` ファイルを使用して、WSSink コンポーネントと AsapSource コンポーネントを関連付けます。 AsapSource は、サーバからデータを読み込み、そのレコードを WSSink に渡します。WSSink は、これらのレコードを Web サービスにパブリッシュします。別の `WsSource.props` ファイルが WSSource コンポーネントと FileSink コンポーネントを関連付けます。 WSSource は、Web サービスにパブリッシュされたレコードを読み込み、FileSink に渡します。FileSink は、これらのレコードをファイルに書き込みます。

1. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`
Windows	コマンド・ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.bat` 2. プロジェクトをクラスタに追加し、クラスタ上で起動。 `start_project.bat`

2. **esp_subscriber** を起動して、クラスタで実行している上記のプロジェクトにサブスクライブします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./esp-subscribe.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `esp-subscribe.bat`

3. **esp_upload** を呼び出して、サーバにレコードをアップロードします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./upload.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **upload.bat**

4. WSSource コンポーネントと FileSink コンポーネント、それらの接続先の Web サービスを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./WsSource.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **WsSource.bat**

5. AsapSource コンポーネントと共に WSSink コンポーネントを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./WsSink.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **WsSink.bat**

この時点で、`out_wssource.txt` ファイルにはレコードがあります。WSSink は、アップロードされたレコードを読み込み、Web サービスに渡します。 WSSource は、これらのレコードを読み込み、FileSink に渡します。FileSink は、これらを `out_wssource.txt` ファイルに書き込みます。

参照：
- *WSSink のプロパティ* (274 ページ)

例：WSSource コンポーネントの使用
soapUI などの Web サービス・クライアントを使用して Web サービスにデータをパブリッシュするには、WSSource コンポーネントを使用します。 WSSource は、Web サービスからレコードを読み込み、FileSink に渡します。FileSink は、これらのレコードをファイルに書き込みます。

1. WSSource が接続されている Web サービスを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./WsSource.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **WsSource.bat**

2. SOAP クライアントを使用して、`examples` フォルダにある `WSSource` フォルダの `readme.txt` ファイルで提供されている呼び出しを試します。 たとえば、SOAP クライアント soapUI を使用します。
   これは、Web サービス・クライアントを使用して、データを Web サービスにパブリッシュします。 WSSource は Web サービスからレコードを読み込み、FileSink に渡します。FileSink はレコードをファイルに書き込みます。

例：XPathMultiTypeXmlReader コンポーネントの使用
XPathMultiTypeXmlReader コンポーネントを、XML フォーマットされたデータを読み込む FilePollSource コンポーネントに関連付け、レコードをサーバにパブリッシュする AsapSink コンポーネントに関連付けます。解析ルールを `XPathXmlStreamReader.props` ファイル内で定義し、これらのルールを使用してファイルから読み込まれる XML レコードを解析します。

1. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`

Windows	コマンド・ウィンドウを開く。
	1. サンプル・クラスタを起動。
	`start_server_cluster.bat`
	2. プロジェクトをクラスタに追加し、クラスタ上で起動。
	`start_project.bat`

2. **esp_subscriber** を起動して、クラスタで実行している上記のプロジェクトにサブスクライブします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./esp-subscribe.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `esp-subscribe.bat`

3. FilePollSource (XPathXmlStreamReader と共に) と AsapSink のコンポーネントを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./XPathMultiTypeXmlReader.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **XPathMultiTypeXmlReader.bat**

`insert.xml` ファイルからのデータがサーバにパブリッシュされます。

参照：

- *XPathMultiTypeXmlReader のプロパティ* (286 ページ)

例： XPathXmlStreamReader コンポーネントの使用
XML データを読み込み、XPath ルールを使用して解析する FilePollSource と共に XPathXmlStreamReader を使用します。次に、レコードをサーバにパブリッシュする AsapSink にレコードを渡します。

1. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`

Windows	コマンド・ウィンドウを開く。
	1. サンプル・クラスタを起動。
	`start_server_cluster.bat`
	2. プロジェクトをクラスタに追加し、クラスタ上で起動。
	`start_project.bat`

2. **esp_subscriber** を起動して、クラスタで実行している上記のプロジェクトにサブスクライブします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./esp-subscribe.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `esp-subscribe.bat`

3. FilePollsource (XPathXmlStreamReader と共に) と AsapSink のコンポーネントを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./XPathXmlStreamReader.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **XPathXmlStreamReader.bat**

`insert.xml` ファイルからのデータがサーバにパブリッシュされます。

参照：
- *XPathXmlStreamReader のプロパティ* (282 ページ)

例：XPathXmlStringWriter コンポーネントの使用
XPathXmlStringWriter コンポーネントは、FileSink コンポーネントと共に使用され、XPath ルールを使用して XML フォーマットされたデータを書き込みます。 サーバからデータを読み込み、FileSink に渡す AsapSource コンポーネントを、XPathXmlStringWriter を使用して XML データを書き出す FileSink コンポーネントに関連付けます。

1. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`

Windows	コマンド・ウィンドウを開く。
	1. サンプル・クラスタを起動。
	`start_server_cluster.bat`
	2. プロジェクトをクラスタに追加し、クラスタ上で起動。
	`start_project.bat`

2. **esp_subscriber** を起動して、クラスタで実行している上記のプロジェクトにサブスクライブします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./esp-subscribe.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `esp-subscribe.bat`

3. AsapSource と FileSink (XPathXmlStringWriter と共に) のコンポーネントを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 **./XPathXmlStringWriter.sh**
Windows	コマンド・ウィンドウを開き、次を入力。 **XPathXmlStringWriter.bat**

`insert.xml` ファイルからのデータがサーバにパブリッシュされます。

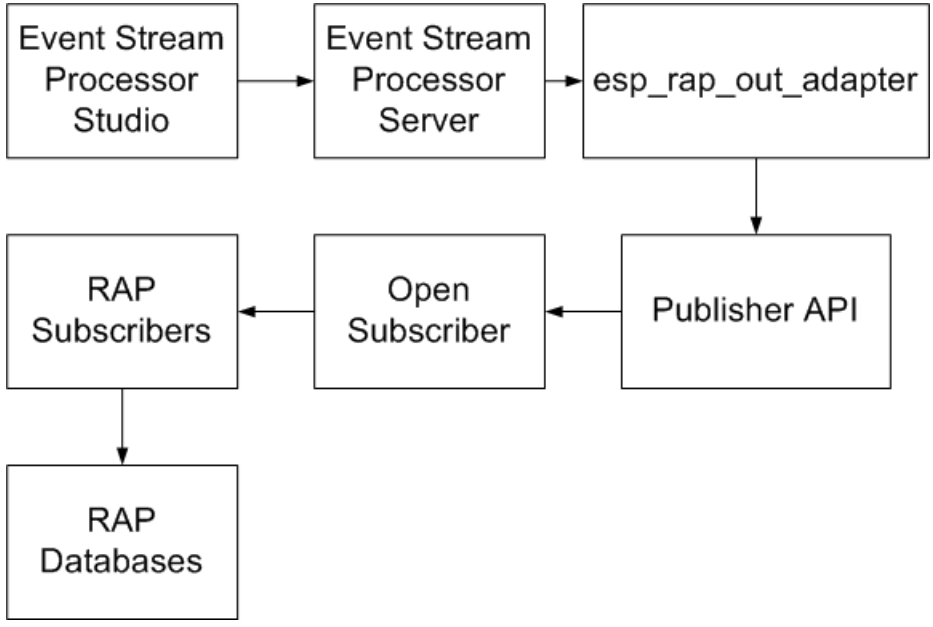
参照：

- *XPathXmlStringWriter のプロパティ* (288 ページ)

RAP アダプタ

アダプタのタイプ： esp_rap_out_adapter。 Sybase Event Stream Processor の RAP アダプタは外部アダプタで、C SDK を使用して、データを Event Stream Processor から RAP プラットフォームにパブリッシュします。

RAP アダプタは、Solaris と Linux のプラットフォームのみをサポートします。



RAP プラットフォームにパブリッシュするストリームごとに、独自のアダプタが必要です。 たとえば、3 つのストリームを RAP にパブリッシュするには、3 つのアダプタを設定します。 これらのアダプタの起動と停止は、個別に実行します。

注意： RAP は入力として挿入のみを受け付けるので、サーバからの削除は削除され、更新は挿入に変換されます。

start コマンド

start.sh スクリプトを使用して、データをサーバから RAP にパブリッシュします。

構文

`start.sh` スクリプトを使用するには、プラットフォーム固有の `libpublisher.so` を `$ESP_HOME/adapters/rap_out/lib` ディレクトリにコピーします。

注意： `start.sh` は、**esp_rap_out_adapter** コマンドの実装です。

```
esp_rap_out_adapter -f configFile -t templateDir -p
publisherConfigDir &
```

注意： `libodbc.so` をインストールします (存在しない場合)。 `libodbc.so` バージョン 1.0.0 へのシンボリック・リンクをファイル名 `libodbc.so.1` で作成し、`$ESP_HOME/adapters/rap_out/lib.` に配置する必要があります。

必須の引数

引数	説明
`-f configFile`	RAP へのデータを提供する Event Stream Processor からのストリームを指定する設定ファイルのフル・パス。 デフォルト値は `../config/espfeedhandler.xml` です。
`-t templateDir`	Event Stream Processor ストリームのカラムを RAP のテーブルとカラムにマップする RDS テンプレートがあるディレクトリへのフル・パス。 デフォルト値は `../templates` です。
`-p PublisherConfigDir`	RAP サブスクライバによって使用されるマルチキャスト・アドレスを定義するパブリッシャ設定ファイルがあるディレクトリへのフル・パス。 デフォルト値は `../config` です。
`&`	(オプション) バックグラウンドでアダプタを実行。

参照：

- *RAP アダプタの起動* (348 ページ)

stop コマンド

フォアグラウンドで RAP アダプタを実行している場合、[Ctrl] キーを押しながら [C] キーを押してこのアダプタを停止できます。

バックグラウンドで RAP アダプタを実行している場合、コマンド・ラインに `ps -eaf | grep esp_rap_out` を入力してアダプタのプロセス ID を取得し、`kill -ll processID` を入力することで、このアダプタを停止できます。

参照：

- *RAP アダプタの停止* (348 ページ)

RAP アダプタのデータ型マッピング

Event Stream Processor のデータ型は、RAP、ASE、IQ のデータ型にマップされます。

RAP アダプタは、Event Stream Processor の `binary` データ型と `boolean` データ型をサポートしていません。

ESP データ型	RAP データ型	ASE データ型	IQ データ型
`integer`	`sint32`	`int`	`int`
`long`	`sint64`	`bigint`	`bigint`
`float`	`decimal(p,s)`	`decimal(p,s)` または `numeric(p,s)`	`decimal(p,s) or numeric(p,s)`
`interval`	`sint64`	`bigint`	`bigint`
`date`	`datetime`	`datetime`	`timestamp`
`timestamp`	`datetime`	`datetime`	`timestamp`
`bigdatetime`	`datetime2`	`bigdatetime`	`timestamp`
`money`	`decimal(19,4)`	`numeric(19,4)`	`numeric(19,4)`
`money(1)`	`decimal(19,1)`	`decimal(19,1)`	`decimal(19,1)`
`money(2)`	`decimal(19,2)`	`decimal(19,2)`	`decimal(19,2)`
`money(3)`	`decimal(19,3)`	`decimal(19,3)`	`decimal(19,3)`
`money(4)`	`decimal(19,4)`	`decimal(19,4)`	`decimal(19,4)`
`money(5)`	`decimal(19,5)`	`decimal(19,5)`	`decimal(19,5)`

ESP データ型	RAP データ型	ASE データ型	IQ データ型
money(6)	decimal(19,6)	decimal(19,6)	decimal(19,6)
money(7)	decimal(19,7)	decimal(19,7)	decimal(19,7)
money(8)	decimal(19,8)	decimal(19,8)	decimal(19,8)
money(9)	decimal(19,9)	decimal(19,9)	decimal(19,9)
money(10)	decimal(19,10)	decimal(19,10)	decimal(19,10)
money(11)	decimal(19,11)	decimal(19,11)	decimal(19,11)
money(12)	decimal(19,12)	decimal(19,12)	decimal(19,12)
money(13)	decimal(19,13)	decimal(19,13)	decimal(19,13)
money(14)	decimal(19,14)	decimal(19,14)	decimal(19,14)
money(15)	decimal(19,15)	decimal(19,15)	decimal(19,15)
string	string	varchar(n)	varchar(n)

設定

RAP アダプタに関する設定情報。

RAP アダプタを設定するには、以下が必要です。

- アダプタ設定ファイル
- パブリッシャ・ファイル
- RDS テンプレート・ファイル

アダプタ設定ファイル

`espfeedhandler.xml` 設定ファイルを使用して、RAP にデータを提供する Event Stream Processor ストリームを指定します。

構文

```
<?xml version="1.0" encoding="UTF-8"?>
<ESPFeedHandler>
    <Logger>
        <LogLevel>warning</LogLevel>
        <LogFile>ESPFeedHandler.log</LogFile>
    </Logger>
    <MainCommandControlServer>
    <MainCCHost>127.0.0.1</MainCCHost>
    <MainCCPort>55555</MainCCPort>
    <Workspace>workspace1</Workspace>
    <Project>project1</Project>
        </MainCommandControlServer>
    <StandbyCommandControlServer>
```

```
        <StandbyCCHost/>
        <StandbyCCPort/>
    </StandbyCommandControlServer>
    <UseEncryption/>
    <ESPAuthentication>
        <User></User>
        <Password></Password>
    </ESPAuthentication>
    <Subscription>        <ProjectionSQL></ProjectionSQL>
<SubscriptionStream>ds1</SubscriptionStream>
        <RAPMessageType>69</RAPMessageType>
    </Subscription>
</ESPFeedHandler>
```

表 9 : XML 要素

要素	説明
ESPFeedHandler	(必須) ファイルのルート要素。
Logger	(必須) ログ記録のアクティビティ設定のルート要素。
LogLevel	(必須) ログ記録のレベル。 有効な値は次のとおりです。 • `error` – エラーのみをログに記録する。 • `warning` - エラーと警告をログに記録する。 • `info` - 情報メッセージと警告レベルのメッセージをログに記録する。 • `debug` - デバッグ・メッセージと情報レベルのメッセージをログに記録する。
LogFile	(必須) ログ・ファイルの名前と場所 (相対パスまたは絶対パス)。
MainCommandControlServer	メイン・コマンド制御サーバの接続情報のルート要素。
MainCCHost	(必須) メイン・コマンド制御サーバの IP アドレス。
MainCCPort	(必須) メイン・コマンド制御サーバのポート。
Workspace	(必須) ストリームがあるクラスタのワークスペース。
Project	(必須) ストリームがあるプロジェクト。
StandbyCommandControlServer	(オプション) スタンバイ・コマンド制御サーバの接続情報のルート要素。
StandbyCCHost	(必須) スタンバイ・コマンド制御サーバの IP アドレス。
StandbyCCPort	(必須) スタンバイ・コマンド制御サーバのポート。

要素	説明
UseEncryption	(オプション) RAP アダプタと Event Stream Processor の間の通信に関する SSL 暗号化のルート要素。
ESPAuthentication	(必須) コマンド制御サーバへの接続を認証するために必要な情報のルート要素。
User	(必須) コマンド制御サーバへの接続に使用するユーザ名。
Password	(必須) コマンド制御サーバへの接続に使用するパスワード (暗号化された形式)。
Subscription	(必須) ストリームへのサブスクリプションに関する情報のルート要素。
ProjectionSQL	(オプション) ストリームで使用する SQL クエリの射影。
SubscriptionStream	(オプション) サブスクライブするストリームの名前。
RAPMessageType	(必須) RAP メッセージ・タイプ番号。 これは、RDS テンプレートで使用される番号と同じです。

パブリッシャ・ファイル

`publisher.xml` ファイルは、Event Stream Processor の RAP パブリッシャを設定します。 このファイルは、ログ・ファイル、管理チャネル、データ・ストリーム・チャネルを指定します。

RAP パブリッシャの設定方法の詳細については、『RAP - The Trading Edition R4.1 Operations Console ユーザーズ・ガイド』の「パブリッシャの設定」を参照してください。

構文

```
<?xml version=”1.0” encoding=”UTF-8”?>
<Publisher>
<Logger>
    <LogLevel>...</LogLevel>
    <LogFile>...</LogFile>
</Logger>
<NumMessageBuffers>...</NumMessageBuffers>
<NumPacketBuffers>...</NumPacketBuffers>
<MessageFlushInterval>...</MessageFlushInterval>
<LatencyCheckInterval>...</LatencyCheckInterval>
<AdminChannel>
 <LocalInterface>...</<LocalInterface>
    <AdminPort>...</AdminPort>
    <MaxConnections>...</MaxConnections>
</AdminChannel>
```

```
<ResendChannel>
    <ResendPort>...</ResendPort>
</ResendChannel>
<TimeToLive>...</TimeToLive>
<DataStreamChannelList>
    <DataStreamChannel>
        <ChannelName>...</ChannelName>
        <LocalInterface>...</LocalInterface>
        <IPAddress>...</IPAddress>
        <Port>...</Port>
    </DataStreamChannel>
    <DataStreamChannel>
        <ChannelName>...</ChannelName>
        <LocalInterface>...</LocalInterface>
        <IPAddress>...</IPAddress>
        <Port>...</Port>
    </DataStreamChannel>
</DataStreamChannelList>
</Publisher>
```

表 10 : XML 要素

要素	説明
Publisher	設定ファイルのルート要素。
Logger	ログ記録のアクティビティに関する設定が含まれています。
LogLevel	ログ記録のレベル。有効な値は次のとおりです。 • Error – エラーのみをログに記録する。 • Warning – エラーに加えて警告もログに記録する。 • Info – 警告レベルのメッセージに加えて情報メッセージもログに記録する。 • Debug – 情報レベルのメッセージに加えてデバッグ・メッセージもログに記録する。
LogFile	ログ・ファイルの名前と場所。 ファイル名には相対パスまたは絶対パスを指定できます。
NumMessageBuffers	メッセージ・バッファの数。 同時に構築されるメッセージ 1 つにつき、メッセージ・バッファが 1 つ必要です。設定できる値は 1 ～ 65535 ですが、使用するマシンに指定バッファ数を保持できるだけのメモリが搭載されている必要があります。

要素	説明
NumPacketBuffers	サブスクライバからのパケット再送信要求を満たすために キャッシュに入れることのできる最大パケット数。 キャッシュされるパケットの数は、1 データ・ストリーム・チャネルあたりの数です。 設定できる値の範囲は 1 ～ 4294967296 ですが、指定したパケット数を保持できるだけのメモリが使用するマシンに搭載されている必要があります。 バッファの数は、 パブリッシャの初期化時に割り付けられます。
MessageFlushInterval	メッセージ・バッファが満たされ始めてからネットワーク上に送信される までの間隔 (秒単位)。1 ～ 65535 の値を 設定できます。
LatencyCheckInterval	メッセージに対して遅延時間チェックを実行した後の秒数。 1 ～ 65535 の値を 設定できます。
AdminChannel	管理チャネルに関する 情報。管理チャネルは、バージョン情報、統計、シャットダウンに関連した要求を 受け入れます。
LocalInterface	パブリッシャでの管理要求のモニタのために使用されるローカル・インタフェース。
AdminPort	UAF エージェントがパブリッシャとの 通信に使用するポート番号。パブリッシャは、このポートに 受信管理要求が送信されてこないかどうかを監視します。
MaxConnections	AdminChannel への同時接続数を決定する。 1 ～ 65535 の範囲の値を指定する必要があります。 デフォルト値は 10 です。
ResendChannel	再送信チャネルに関する情報。 このチャネルで、サブスクライバから接続要求が送信されてこないかどうかを監視します。サブスクライバは、パブリッシャへの接続を開いてパケットの再送信要求を 発行します。
ResendPort	サブスクライバが 、欠落したネットワーク・パケットの再送信を要求するため、およびパブリッシャとサブスクライバとの間のネットワーク遅延時間を 測定するために使用するポートの番号。
TimeToLive	メッセージが期限切れになる前に渡すことのできるルーティング・デバイス数の制限。
DataStreamChannelList	データ・ストリーム・チャネル定義のリスト。 最大 255 個のデータ・ストリーム・チャネルが定義されます。

要素	説明
DataStreamChannel	単一のデータ・ストリーム・チャネルに関する情報が 含まれています。パブリッシャが送信する各メッセージは、定義されたチャネルの 1 つを経由して ネットワーク・パケット・バッファに送信されます。パブリッシャは、システムに 負荷がかかっている間、チャネルすべてにバランス良くメッセージを 振り分けます。
ChannelName	チャネルの詳細な名前。ログを記録するときのチャネルの特定に使用されます。
LocalInterface	データ送信に使用されるローカル・マシン上の ネットワーク・インタフェースの IP アドレス。
IPAddress	ネットワークを介してメッセージを 送信するための UDP マルチキャスト・アドレス。
Port	UDP マルチキャストを使用してメッセージを 送信するときに使用するポート。

RDS テンプレート・ファイル

RAP データ・ストリーム (RDS) テンプレート・ファイルは、Event Stream Processor での RAP メッセージ・タイプの構造を定義します。

RAP のメッセージとスキーマの詳細については、『RAP - The Trading Edition R4.1 開発者ガイド』の「RAP メッセージとスキーマのカスタマイズ」の章を参照してください。

構文

```
<?xml version="1.0" encoding="UTF-8"?>

<Template xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="../template.xsd">
    <MessageDefnList>
        <MessageDefn>
            <MessageDesc>...</MessageDesc>
            <MessageType>...</MessageType>
            <DestTableName>...</DestTableName>
            <FieldDefnList>
                <FieldDefn>
                    <FieldName>...</FieldName>
                    <StringField/>
                    <DestColumnName>...</DestColumnName>
                    <Lookup>
                        <LookupTableName>...T</LookupTableName>
                        <LookupColumnName>...</LookupColumnName>
                        <LookupColumnReturn>...</LookupColumnReturn>
```

```
                    </Lookup>
                </FieldDefn>
            </FieldDefnList>
        </MessageDefn>
    </MessageDefnList>
</Template>
```

表 11 : XML 要素

要素	説明
Template	テンプレートのルート要素。
MessageDefnList	1 つ以上のメッセージ定義から 成るリスト。
MessageDefn	単一のメッセージ・タイプを 定義する情報。
MessageDesc	たとえば、株式相場などの 市場データ・メッセージのタイプの説明。この要素は、説明の目的でのみ 使用され、任意の文字列を設定できます。
MessageType	市場データ・メッセージのタイプ を表す一意の番号。この番号は、全テンプレート内のすべてのメッセージ定義にわたってメッセージ・ タイプを一意である必要があります。値には 1 ～ 65535 の間の任意の整数を設定できます。
DestTableName	メッセージの格納先とする データベース・テーブルの名前。メッセージ・ タイプ 1 つにつきデータベース・テーブルは 1 つあります。値には任意の文字列を含めることができます。
FieldDefnList	1 つ以上のフィールド定義から 成るリスト。
FieldDefn	単一のフィールドを定義する 情報。
FieldName	フィールドの名前。値には任意の文字列を 含めることができます。
IntegerField	このフィールドが integer データ型の 1 つであることを示す。
IntegerDataType	整数フィールドのデータ型。有効な 値は、 `uint8`、`uint16`、`uint32`、`uint64`、`sint8`、`sint16`、`sint32`、または `sint64` のいずれかです。
DecimalField	このフィールドが 10 進数値であることを 示します。 フィールド定義には、IntegerField、DecimalField、StringField、DateField、TimeField、DateTimeField、DateTime2Field、または Time2Field のいずれか 1 つだけがあります。
Precision	10 進数フィールドの精度。使用可能な最大精度は、38 桁です。

要素	説明
Scale	10 進数フィールドの位取り (小数点以下の桁数)。最大の位取りは、精度 (38) 以下です。
StringField	このフィールドが string データ型であることを示す。
DateField	このフィールドが date データ型であることを示す。 フィールド定義には、IntegerField、DecimalField、StringField、DateField、TimeField、DateTimeField、DateTime2Field、または Time2Field のいずれか 1 つだけがあります。
TimeField	このフィールドが time データ型であることを示す。
Time2Field	このフィールドが bigtime データ型 (小数第 6 位まで格納) であることを示す。 フィールド定義には、IntegerField、DecimalField、StringField、DateField、TimeField、DateTimeField、DateTime2Field、または Time2Field のいずれか 1 つだけがあります。
DateTimeField	このフィールドが datetime データ型であることを示す。
DateTime2Field	このフィールドが bigdatetime 値 (小数第 6 位まで格納) であることを示します。 フィールド定義には、IntegerField、DecimalField、StringField、DateField、TimeField、DateTimeField、DateTime2Field、または Time2Field のいずれか 1 つだけがあります。
DestColumnName	フィールド・データの格納先 とするカラムの名前。フィールド 1 つにつきカラムは 1 つあります。この 要素の値には任意の文字列を含めることができます。
Lookup	(オプション) このフィールドのデータを別のテーブルの検索として使用する必要があることを示す。 この要素がフィールド定義に含まれていない場合、検索は必要ありません。
LookupTableName	値を検索するために使用される テーブルの名前。
LookupColumnName	値を検索するために使用される カラムの名前。
LookupColumnReturn	検索の戻り値データを 出力するカラムの名前。

例：RAP アダプタの設定

RAP と Event Stream Processor との間の通信用に RAP アダプタを設定するには、設定、パブリッシャ、RDS テンプレート・ファイルを設定します。

1. `$RAPOUT_HOME` 環境変数を `$ESP_HOME/adapters/rap_out` ディレクトリに設定します。
2. `$RAPOUT_HOME` ディレクトリに移動します。
3. RAP にパブリッシュするストリームを定義するプロジェクトを作成し、`$ESP_HOME/bin` ディレクトリのファイル `model.ccl` に保存します。
4. Event Stream Processor を起動します。

   Windows：

   1. サンプル・クラスタを起動します。

      ```
      cd %ESP_HOME%¥cluster¥nodes¥node1
        %ESP_HOME%¥bin¥esp_server --cluster-node node1.xml
      ```

   2. CCL をコンパイルして CCX を作成します。

      ```
      %ESP_HOME%¥bin¥esp_compiler -i model.ccl -o model.ccx
      ```

   3. プロジェクトをクラスタに展開します。

      ```
      %ESP_HOME%¥bin¥esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --add_project --workspace-
      name=w1 --project-name=p1 --ccx=model.ccx
      ```

   4. 展開したクラスタをサーバで起動します。

      ```
      %ESP_HOME%¥bin¥esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --start_project --
      workspace-name=w1 --project-name=p1
      ```

   UNIX：

   1. サンプル・クラスタを起動します。

      ```
      cd $ESP_HOME/cluster/nodes/node1
        $ESP_HOME/bin/esp_server --cluster-node node1.xml
      ```

   2. CCL をコンパイルして CCX を作成します。

      ```
      $ESP_HOME/bin/esp_compiler -i model.ccl -o model.ccx
      ```

   3. プロジェクトをクラスタに展開します。

      ```
      $ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --add_project --workspace-
      name=w1 --project-name=p1 --ccx=model.ccx
      ```

   4. 展開したクラスタをサーバで起動します。

      ```
      $ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --start_project --
      workspace-name=w1 --project-name=p1
      ```

5. `$RAPOUT_HOME/config` ディレクトリで、`espfeedhandler.xml` 設定ファイルを変更し、RAP にデータを提供する Event Stream Processor ストリームを指

定します。 たとえば、次のファイルは、1 つのストリーム Trades をパブリッシュするようにアダプタを設定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<ESPFeedHandler>
        <Logger>
                <LogLevel>warning</LogLevel>
                <LogFile>ESPFeedHandler.log</LogFile>
        </Logger>
        <MainCommandControlServer>
       <MainCCHost>127.0.0.1</MainCCHost>
       <MainCCPort>19011</MainCCPort>
       <Workspace>w1</Workspace>
       <Project>p1</Project>
        </MainCommandControlServer>
        <StandbyCommandControlServer>
                <StandbyCCHost/>
                <StandbyCCPort/>
        </StandbyCommandControlServer>
        <UseEncryption/>
        <ESPAuthentication>
                <User></User>
                <Password></Password>
        </ESPAuthentication>
        <Subscription>
                <ProjectionSQL></ProjectionSQL>
                <SubscriptionStream>Trades</SubscriptionStream>
                <RAPMessageType>69</RAPMessageType>
        </Subscription>
</ESPFeedHandler>
```

6. `$RAPOUT_HOME/config` で、既存のパブリッシャ・ファイルを変更して、RAP サブスクライバによって使用されるマルチキャスト・アドレスを指定します。例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<Publisher>
        <Logger>
                <LogLevel>debug</LogLevel>
                <LogFile>Publisher.log</LogFile>
        </Logger>
    <NumMessageBuffers>1</NumMessageBuffers>
    <NumPacketBuffers>10000</NumPacketBuffers>
        <MessageFlushInterval>1</MessageFlushInterval>
        <LatencyCheckInterval>30</LatencyCheckInterval>
        <AdminChannel>
        <LocalInterface>testmachine</LocalInterface>
        <AdminPort>5002</AdminPort>
        </AdminChannel>
        <ResendChannel>
        <ResendPort>5103</ResendPort>
        </ResendChannel>

        <TimeToLive>1</TimeToLive>

        <DataStreamChannelList>
```

```
        <DataStreamChannel>
            <ChannelName>test2</ChannelName>
            <LocalInterface>127.0.0.1</LocalInterface>
            <IPAddress>224.0.2.0</IPAddress>
            <Port>5050</Port>
        </DataStreamChannel>
        <DataStreamChannel>
            <ChannelName>test1</ChannelName>
            <LocalInterface>127.0.0.1</LocalInterface>
            <IPAddress>224.0.2.0</IPAddress>
            <Port>5800</Port>
        </DataStreamChannel>
        </DataStreamChannelList>
</Publisher>
```

7. `$RAPOUT_HOME/templates` で、RAP にパブリッシュするストリームごとに RDS テンプレートを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>

<Template xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="../template.xsd">
    <MessageDefnList>
        <MessageDefn>
            <MessageDesc>Split Event</MessageDesc>
            <MessageType>70</MessageType>
            <DestTableName>rapout2</DestTableName>
            <FieldDefnList>
                <FieldDefn>
                    <FieldName>integer</FieldName>
                    <IntegerField>
                        <IntegerDataType>sint32</IntegerDataType>
                    </IntegerField>
                    <DestColumnName>int16</DestColumnName>
                </FieldDefn>
        <FieldDefn>
            <FieldName>string</FieldName>
            <StringField/>
            <DestColumnName>string</DestColumnName>
        </FieldDefn>
                <FieldDefn>
                    <FieldName>int32</FieldName>
                    <IntegerField>
                        <IntegerDataType>sint32</IntegerDataType>
                    </IntegerField>
                    <DestColumnName>int32test</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>int64</FieldName>
                    <IntegerField>
                        <IntegerDataType>sint64</IntegerDataType>
                    </IntegerField>
                    <DestColumnName>int64test</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>double</FieldName>
```

```
                        <DecimalField>
                            <Precision>18</Precision>
                            <Scale>4</Scale>
                        </DecimalField>
                    <DestColumnName>doubletest</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>money</FieldName>
                    <DecimalField>
                        <Precision>18</Precision>
                        <Scale>4</Scale>
                    </DecimalField>
                    <DestColumnName>money_test</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>money(1)</FieldName>
                    <DecimalField>
                        <Precision>18</Precision>
                        <Scale>1</Scale>
                    </DecimalField>
                    <DestColumnName>money1</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>money(2)</FieldName>
                    <DecimalField>
                        <Precision>18</Precision>
                        <Scale>2</Scale>
                    </DecimalField>
                    <DestColumnName>money2</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>money(3)</FieldName>
                    <DecimalField>
                        <Precision>18</Precision>
                        <Scale>3</Scale>
                    </DecimalField>
                    <DestColumnName>money3</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>money(4)</FieldName>
                    <DecimalField>
                        <Precision>18</Precision>
                        <Scale>4</Scale>
                    </DecimalField>
                    <DestColumnName>money4</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>money(5)</FieldName>
                    <DecimalField>
                        <Precision>18</Precision>
                        <Scale>5</Scale>
                    </DecimalField>
                    <DestColumnName>money5</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>money(6)</FieldName>
```

```
                    <DecimalField>
                        <Precision>18</Precision>
                        <Scale>6</Scale>
                    </DecimalField>
                    <DestColumnName>money6</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>money(7)</FieldName>
                    <DecimalField>
                        <Precision>18</Precision>
                        <Scale>7</Scale>
                    </DecimalField>
                    <DestColumnName>money7</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>money(8)</FieldName>
                    <DecimalField>
                        <Precision>18</Precision>
                        <Scale>8</Scale>
                    </DecimalField>
                    <DestColumnName>money8</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>money(9)</FieldName>
                    <DecimalField>
                        <Precision>18</Precision>
                        <Scale>9</Scale>
                    </DecimalField>
                    <DestColumnName>money9</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>money(10)</FieldName>
                    <DecimalField>
                        <Precision>18</Precision>
                        <Scale>10</Scale>
                    </DecimalField>
                    <DestColumnName>money10</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>money(11)</FieldName>
                    <DecimalField>
                        <Precision>18</Precision>
                        <Scale>11</Scale>
                    </DecimalField>
                    <DestColumnName>money11</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>money(12)</FieldName>
                    <DecimalField>
                        <Precision>18</Precision>
                        <Scale>12</Scale>
                    </DecimalField>
                    <DestColumnName>money12</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>money(13)</FieldName>
```

```
                    <DecimalField>
                        <Precision>18</Precision>
                        <Scale>13</Scale>
                    </DecimalField>
                    <DestColumnName>money13</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>money(14)</FieldName>
                    <DecimalField>
                        <Precision>18</Precision>
                        <Scale>14</Scale>
                    </DecimalField>
                    <DestColumnName>money14</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>money(15)</FieldName>
                    <DecimalField>
                        <Precision>18</Precision>
                        <Scale>15</Scale>
                    </DecimalField>
                    <DestColumnName>money15</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>interval</FieldName>
                    <IntegerField>
                        <IntegerDataType>sint64</IntegerDataType>
                    </IntegerField>
                    <DestColumnName>interval</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>bigdatetime</FieldName>
                    <DateTime2Field/>
                    <DestColumnName>bigdatetime</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>date</FieldName>
                    <DateTimeField/>
                    <DestColumnName>date_test</DestColumnName>
                </FieldDefn>
                <FieldDefn>
                    <FieldName>timestamp</FieldName>
                    <DateTimeField/>
                  <DestColumnName>timestamp_test</DestColumnName>
                </FieldDefn>
    </FieldDefnList>
        </MessageDefn>
    </MessageDefnList>
</Template>
```

テンプレート・ファイルが、RAP サブスクライバ・テンプレート・ディレクトリにコピーされていることを確認します。

オペレーション

RAP アダプタの起動と停止は、コマンド・ラインから実行します。

RAP アダプタの起動
アダプタの設定が完了したら、**start.sh** スクリプトを使用してアダプタを起動します。

前提条件

- RAP データベース (データベース内にメッセージ・テーブルが存在します)、RAP サブスクライバ、サーバ、アダプタを接続するプロジェクトを起動する。
- 存在しない場合、`libodbc.so` をインストールする。 `libodbc.so` バージョン 1.0.0 へのシンボリック・リンクをファイル名 `libodbc.so.1` で作成し、`$ESP_HOME/adapters/rap_out/lib.` に配置する必要があります。
- `start.sh` スクリプトを使用するには、プラットフォーム固有の `libpublisher.so` を `$ESP_HOME/adapters/rap_out/lib` ディレクトリにコピーします。

手順

1. RAP OpsConsole で `start` を選択し、RAP データベース (RAPCache と RAPStore) を起動します。
2. RAP OpsConsole で `start` を選択し、RAP サブスクライバを起動します。
3. コマンド・プロンプトで **start.sh** スクリプトを実行します。

   start.sh スクリプトで次を実行します。

   ```
   esp_rap_out_adapter -f $RAPOUT_HOME/config/espfeedhandler.xml -t
   $RAPOUT_HOME/templates -p $RAPOUT_HOME/config
   ```

参照：

- *start コマンド* (332 ページ)

RAP アダプタの停止
アダプタの設定が完了したら、**esp_rap_out_adapter** コマンドを使用してアダプタを停止します。

1. クラスタをシャットダウンします。
   - アダプタをフォアグラウンドで実行している場合、アダプタを起動したウィンドウに移動し、[Ctrl] キーを押しながら [C] キーを押す。
   - バックグラウンドで RAP アダプタを実行している場合、`ps -eaf | grep esp_rap_out_adapter` を入力してアダプタのプロセス ID を取得し、`kill -ll processID` を入力する。

2. RAP OpsConsole で `stop` を選択し、RAP サブスクライバをシャットダウンします。
3. 以下を実行して、RAP データベースをシャットダウンします。
   - RAP OpsConsole で `stop` を選択する。
   - バックグラウンドで RAP アダプタを実行している場合、`ps -eaf | grep dataserver` (RAPCache の場合) または `ps -eaf | grep IQSRV15` (RAPStore の場合) を入力してアダプタのプロセス ID を取得し、`kill -ll processID` を入力する。

参照：
- *stop コマンド* (333 ページ)

ロイター・マーケットフィード・アダプタ

Sybase ロイター・マーケットフィード・アダプタは、Event Stream Processor と Reuters Market Data System (RMDS) との間のソフトウェア・インタフェースです。このアダプタは、ロイター・マーケットフィードのメッセージ・フォーマットを使用します。

アダプタは、入力アダプタまたは出力アダプタとして設定できます。 入力アダプタは、RMDS 上の 1 つ以上の Reuters Instrument Code (RIC) にサブスクライブし、入力を Event Stream Processor に提供します。 出力アダプタは、Event Stream Processor からの出力を RMDS にパブリッシュします。 これによって、Event Stream Processor は Reuters のインフラストラクチャが提供するスピードと信頼性を活用して、データを配信できます。

ロイター・マーケットフィード・インプット・アダプタは、スキーマ検出をサポートします。 入力と出力の機能を必要とする場合は、2 つのアダプタ・インスタンスを実行します。

アダプタは、Solaris™ と Linux® のオペレーティング・システム上でのみ実行しますが、Solaris、Linux、または Windows® 上で実行する Event Stream Processor ソフトウェアと共に使用できます。

稼働条件

ロイター・マーケットフィード・インプット・アダプタとロイター・マーケットフィード・アウトプット・アダプタには、いくつかの要件があります。

入力アダプタは、次を必要とします。

- マーケットフィード・プロトコルを使用する RMDS マーケット・データ接続
- 1 つ以上の金融商品のデータに対する有効なサブスクリプション

出力アダプタは、次を必要とします。

- マーケットフィード・プロトコルを使用してデータを RMDS に送信することをサポートする有効な接続

一般的な考慮事項

ロイター・マーケットフィード・アダプタを実行する各ユーザ・アカウントにユーザがアクセスできるようにし、Reuters からの入力接続と Reuters への出力接続を設定します。

ユーザ・アクセスの有効化

ロイター・マーケットフィード・アダプタを使用する各ユーザ・アカウントにユーザがアクセスできるようにします。

1. ユーザ・アカウントが、インストールされたソフトウェアを実行するパーミッションを持っていることを確認します。
2. 環境変数 $ESP_REUTERS_HOME を追加し、アダプタ階層のルート (ユーザの実行時環境) に設定します。
3. (オプション) 環境変数をシェル・プロファイルに追加します。
4. Event Stream Processor は、RSA、LDAP、Kerberos の認証をサポートします。インストール環境でこれらの認証方法のいずれかが使用される場合、その認証方法で動作するようにユーザ・アカウントがセットアップされていることを確認します。

Reuters からの入力接続の設定

サンプル設定ファイルを、サイトの RMDS 接続用に変更します。 複数の RMDS 接続を使用するアダプタが複数ある場合、それぞれに個別の一意な名前付き設定ファイルを使用する必要があります。 異なる名前の設定ファイルについて、入力アダプタ・マップ・ファイルのエントリを変更するか、そのファイル名を **esp_rmds** コマンドの -f オプションを使用して指定します。

前提条件

- サイト固有の設定ファイルを格納するディレクトリを作成 (または選択) する。
- 環境変数 (MY_CONFIG) を作成し、そのディレクトリのフル・パス名に設定する。

手順

インストール・プロセス時に、サンプル設定ファイル (`rfasub.cfg`) が $ESP_REUTERS_HOME/config ディレクトリに格納されます。 このファイルは、以下に示すように、設定ファイル用の Reuters フォーマットに従います。

```
# Change this if necessary.
# the port number of the P2PS (default 8101)
¥Connections¥Connection_SSLED¥PortNumber          = 8101
```

```
# Change this if necessary.
# the user name to connect with (should be the DACS name if DACS is
enabled)
¥Connections¥Connection_SSLED¥UserName             = "triarch"

# Change this if necessary.
# a list of P2PS host names
¥Connections¥Connection_SSLED¥ServerList           = "localhost"

# Refer to RFA documentation for more advanced changes to the
remaining entries
¥Connections¥Connection_SSLED¥connectionType       = "SSLED"

¥Adapters¥SASS3_Adapter¥requestQueueReadThreshold  = 1
¥Adapters¥SASS3_Adapter¥mainLoopTimerInterval      = 200

¥Adapters¥SSLED_Adapter¥masterFidFile              = "config/
appendix_a"
¥Adapters¥SSLED_Adapter¥enumTypeFile               = "config/
enumtype.def"
¥Adapters¥SSLED_Adapter¥downloadDataDict           = false

# Change the fileLoggerFilename appropriately for your setup
¥Logger¥AppLogger¥windowsLoggerEnabled             = false
¥Logger¥AppLogger¥fileLoggerEnabled                = true
¥Logger¥AppLogger¥fileLoggerFilename              = "rfasub.{p}.log"

¥Control¥Entitlements¥dacs_SbeEnabled              = false
¥Control¥Entitlements¥dacs_CbeEnabled              = false

¥Logger¥ComponentLoggers¥Connections¥messageFile   = "config/
messages/RFA7_Connections.mc"
¥Logger¥ComponentLoggers¥Adapter¥messageFile       = "config/
messages/RFA7_Adapter.mc"
¥Logger¥ComponentLoggers¥SessionCore¥messageFile   = "config/
messages/RFA7_SessionLayer.mc"
¥Logger¥ComponentLoggers¥SSLED_Adapter¥messageFile = "config/
messages/RFA7_SSLED_Adapter.mc"


¥Sessions¥Session1¥connectionList                   =
"Connection_SSLED"
```

1. 次の情報をシステム管理者から取得します。
   - RMDS からマーケットフィード・データを受信するサーバの名前
   - システムが接続するマシンのポート番号
   - Reuters への接続用に定義されているユーザ名
   - サブスクライブ先の各 Reuters サービスの名前

2. サンプル設定ファイルのコピーを $MY_CONFIG ディレクトリに作成します。
   ```
   cp $ESP_REUTERS_HOME/config/rfasub.cfg $MY_CONFIG
   ```
3. エディタを使用して設定ファイルを開きます。
4. ¥Connections¥Connection_SSLED¥PortNumber 行で、デフォルトのポート番号 (8101) を Reuters 接続で使用される番号に置き換えます (異なる場合)。
5. ¥Connections¥Connection_SSLED¥UserName 行で、triarch を Reuters サブスクリプションのユーザ名に置き換えます。 囲んでいる引用符を維持します。
6. ¥Connections¥Connection_SSLED¥ServerList 行で、localhost を RMDS からマーケットフィード・データを受信するサーバの名前に置き換えます。 囲んでいる引用符を維持します。
   複数のサーバが RMDS からデータを受信している場合、それらの名前をすべて、優先順に基づくカンマ区切りのリストで指定します。
7. (オプション) ¥Logger¥AppLogger¥fileLoggerFilename 行で、ログ・ファイルの名前を変更します。
   ここで指定されているデフォルトのファイル名 rfasub.{p}.log には、文字列 {p} があり、Reuters ライブラリによって、ログ・ファイルの作成時に UNIX プロセス ID に置き換えられます。
8. 変更したファイルを保存します。
   設定ファイルの他のパラメータも、ロイター・マーケットフィード・アダプタの機能に影響を及ぼすので、同様に変更することをおすすめします。

Reuters への出力接続の設定

サンプル設定ファイルを、サイトの RMDS 接続用に変更します。 複数の RMDS 接続を使用するアダプタが複数ある場合、それぞれに個別の一意な名前付き設定ファイルを使用する必要があります。 異なる名前の設定ファイルについて、出力アダプタ・マップ・ファイルのエントリを変更するか、そのファイル名を **esp_rmds** コマンドの -f オプションを使用して指定します。

前提条件

- サイト固有の設定ファイルを格納するディレクトリを作成 (または選択) する。
- 環境変数 (MY_CONFIG) を作成し、そのディレクトリのフル・パス名に設定する。

手順

インストール・プロセス時に、サンプル設定ファイル (rfapub.cfg) が $ESP_REUTERS_HOME/config ディレクトリに格納されます。 このファイルは、以下に示すように、設定ファイル用の Reuters フォーマットに従います。

```
# Change this if necessary.
# This needs to match port number for the route as defined in the
Source Distributor.
¥Connections¥Connection_SSLED_MP¥ipcServerName      = "8105"

# Refer to RFA documentation for more advanced changes to the
remaining entries.
¥Connections¥Connection_SSLED_MP¥connectionType     = "SSLED_MP"
¥Connections¥Connection_SSLED_MP¥entitlementData    = false
¥Sessions¥Session1¥connectionList                   =
"Connection_SSLED_MP"

# Change the fileLoggerFilename appropriately for your setup
¥Logger¥AppLogger¥windowsLoggerEnabled              = false
¥Logger¥AppLogger¥fileLoggerEnabled                 = true
¥Logger¥AppLogger¥fileLoggerFilename                = "./rfapub.
{p}.log"

¥Control¥Entitlements¥dacs_SbeEnabled               = false
¥Control¥Entitlements¥dacs_CbeEnabled               = false

¥Logger¥ComponentLoggers¥Connections¥messageFile      = "./config/
messages/RFA7_Connections.mc"
¥Logger¥ComponentLoggers¥Adapter¥messageFile          = "./config/
messages/RFA7_Adapter.mc"
¥Logger¥ComponentLoggers¥SessionCore¥messageFile      = "./config/
messages/RFA7_SessionLayer.mc"
¥Logger¥ComponentLoggers¥SSLED_Adapter¥messageFile    = "./config/
messages/RFA7_SSLED_Adapter.mc"
¥Logger¥ComponentLoggers¥SSLED_MP_Adapter¥messageFile = "./config/
messages/RFA7_SSLED_MP_Adapter.mc"
```

1. 次の情報をシステム管理者から取得します。
   - src_dist または RMDS インフラストラクチャ・サーバがロイター・マーケットフィード・アダプタからの更新情報を受信するポート番号
   - Event Stream Processor から更新情報を受信するサーバの名前
2. サンプル設定ファイルのコピーを `$MY_CONFIG` ディレクトリに作成します。
   ```
   cp $ESP_REUTERS_HOME/config/rfapub.cfg $MY_CONFIG
   ```
3. エディタを使用して設定ファイルを開きます。
4. `¥Connections¥Connection_SSLED_MP¥ipcServerName` 行で、デフォルトのポート番号 (8105) を、src_dist がロイター・マーケットフィード・アダプタからの更新情報を受信するポート番号に置き換えます (異なる場合)。
5. (オプション) `¥Logger¥AppLogger¥fileLoggerFilename` 行で、ログ・ファイルの名前を変更します。 ここで指定されているデフォルトのファイル

名 `./rfapub.{p}.log` には、文字列 {p} があり、Reuters ライブラリによって、ログ・ファイルの作成時に UNIX プロセス ID に置き換えられます。

6. 変更したファイルを保存します。

入力アダプタの設定

RMDS (Reuters Market Data Service) から Event Stream Processor にデータを送信するように、入力アダプタを設定します。

必要なデータとシステムをセットアップする方法を決定してから、入力アダプタを設定します。

受信データの送信元の Event Stream Processor インスタンスについて、以下の情報を取得する必要があります。

- クラスタ環境で適用可能なセキュリティ・オプションと、ワークスペースとプロジェクトの名前。
- 使用される認証メカニズム (Kerberos、RSA、LDAP、または none)。

データの決定

受信 Reuters データをプロジェクトに取り込む方法を決定します。

また、Level1 データまたは Level2 データを必要とするかどうかを決定します。Level1 データについては、ロイター・マーケットフィード・アダプタを使用し、Level2 データについては、ロイター OMM アダプタを代わりに使用します。

決定事項	説明
取引所	情報を取得する取引所を決定 (たとえば、NYSE、NASDAQ、Toronto など)。
RIC と FID	必要なマーケット・データを決定。 特に、アダプタが Event Stream Processor に提供する RIC (Reuters Instrument Code) と、これらの使用する金融商品タイプのためのロイター Field ID (FID) を決定します。
ストリーム	Reuters アダプタは、Event Stream Processor 上の 1 つ以上のストリームにデータを供給することが可能。 アダプタによって提供されるロイター・マーケット・データを使用するには、どの既存のデータ・ストリームをアダプタのデータ・フィードにマップするかを決定するか、または 1 つ以上の新しいストリームを定義します。

管理上の決定
プロジェクトに関して、いくつかの管理上の決定を行う必要があります。

決定事項	説明
セッション名	プロジェクトとアダプタ・マップ・ファイルを結び付ける任意の文字列。 わかりやすい文字列を使用します。
ロギングとストリーム出力用のディレクトリ	アダプタは、独自のログ・メッセージを書き込み、Reuters ログ・メッセージの個別のセットを生成可能。 設定では、これらのログ・ファイルが書き込まれるかどうかと、どこに書き込まれるかを指定します。
Sybase ユーザ・アカウント	Event Stream Processor の起動時に認証を指定していない場合を除き、アダプタが使用する、有効な Event Stream Processor ユーザ・アカウントを指定。

入力アダプタ・マップ・ファイル
マップ・ファイルは、ロイター・マーケットフィード・アダプタと Event Stream Processor との間のインタフェースを設定します。どのソース・ストリームが RMDS からアダプタを介してデータを受信するかを指定し、特定の RMDS Field ID (FID) をそのソース・ストリームの特定のカラムにマップします。

入力アダプタ・マップ・ファイルを作成するには、次の 3 つの主要なタスクを実行する必要があります。

- Event Stream Processor 設定ファイルで定義されている 1 つ以上のストリームのカラムに、受信データ要素を一致させる。
- アダプタからの各更新情報で提供される RIC を Event Stream Processor 設定ファイルのローと一致させる。
- アダプタからの各更新情報が、ストリームのカラム定義で定義されているように、更新対象の各ストリームの一意なキーを提供するレコードに変換できるようにする。

データ構造
データ構造には、構造上重要な 3 つの側面があります。 データ・カラム、データ型、キー値です。

- 各データ・ストリームは、1 つ以上のデータ・カラムで構成される。
- 各カラムには、データ型が設定されている。
- ほとんどのストリームで、各ローに一意なキー値がある。 ソース・ストリーム定義には、「キー」カラムとして 1 つ以上のカラムが指定されています。

受信 RMDS データ
アダプタが特定の RIC の RMDS にサブスクライブすると、RMDS は、最初に、その RIC の利用可能なすべてのマーケット・データで構成される初期イメージを送信します。

その後、RMDS は、サブスクライブしている RIC のいずれかの値が変更されると更新情報のみを送信します。 各更新情報は、Field ID (FID) で識別される RIC と各変更の新しい値で構成されます。 RMDS 用に定義された各 FID には、データ型が設定されています。

マーケット・データ・フィールド・マッピング
対象の Event Stream Processor ストリームの各カラムをロイター FID または「疑似フィールド」にマップします。

ストリームの各カラムに対応する FID を見つけます。 Event Stream Processor カラムのデータ型は、それをフィードするロイター FID のデータ型と互換性を持つ必要があります。

次に、互換性のある FID のデータ型と Event Stream Processor のデータ型を示します。

Event Stream Processor のデータ型	Reuters のデータ型
`integer` または `long`	`integer`
`string`	`alphanumeric`
`integer`	`enumerated`
`timestamp` または `date`	`time, date`
`money` または `float`	`price`
`long` または `integer`	`time_seconds`
サポートされていません。	`binary`

Reuters Instrument Code マッピング
各受信 RMDS 更新情報の識別子は、RIC (Reuters Instrument Code) です。

RIC をストリームのデータ型 string のカラムにマップします。マップ先のストリームに適切なカラムがない場合は、ストリームにカラムを追加するか、別のストリームにマップします。

ストリームのキーとの一致

アダプタ・マップ・ファイルでは、Event Stream Processor ストリームに送信される各更新情報に、そのストリーム用に定義されている一意なキーに一致するフィールドまたはフィールドの組み合わせがあるようにアダプタを設定する必要があります。 この条件をより柔軟にするために、アダプタ設定メカニズムでは、「疑似フィールド」をサポートしています。

アダプタが RMDS から受信するマーケット・データの更新情報は、マップ・ファイルの dataField 要素または dateTimeField 要素を使用して Event Stream Processor ストリームのカラムにマップされます。 RMDS には、マーケット・データ以外の情報もあり、各更新情報には RIC があります。 さらに、各更新情報にシーケンス番号を付加するようにアダプタを構成できます。

これらのデータ項目をマッピング・プロセスで利用できるようにするために、マップ・ファイル・メカニズムは、「疑似フィールド」と呼ばれるこれらの要素をサポートします。

フィールド	説明	データ型
`itemName`	RIC。	`string`(必須)
`serviceName`	RMDS によって受信された、この RIC からのマーケット・データの送信元サービスの名前。	`string` (オプション)
`itemStale`	項目の状態。	`integer` (オプション)
`sequenceNumber`	アダプタによって各受信イベントに順次に割り当てられる一意な番号 (更新を発生させるかどうかに関係なし)。	`long` (オプション)
`FIDListField`	更新情報内の更新された各値の FID 名と値。	`string` (オプション)
`updateNumber`	アダプタによって各受信更新情報に順次に割り当てられる一意な番号。	`long` (オプション)

プロジェクトからストリーム情報の取得

Reuters ストリームについて必要な情報を収集します。

入力アダプタを設定する場合は、最初に、RMDS マーケット・データを受信する、Event Stream Processor 上のソース・ストリームを決定します。 この目的のスト

リームが Event Stream Processor プロジェクトにない場合は、Reuters アダプタと共に使用する新しい 1 つ以上のストリームを定義します。

ロイター・マーケットフィード・アダプタからデータを受信するストリームを選択 (または定義) したら、そのストリームについての情報をプロジェクト・ファイルから収集します。 Event Stream Processor プロジェクト・ファイルには、1 つ以上のストリーム定義があります。 各ストリーム定義は、Event Stream Processor の起動時にインスタンス化されるデータ・ストリームを指定します。 ストリーム定義は、以下で構成されます。

- ストリームの一意な ID
- ストリーム・データ用のデータベース・ストアと出力ファイル
- データ・ストリーム内の各ローの一意なキー値として使用されるカラムのリスト

Reuters アダプタによって提供される RMDS データを転送するストリームを決定したら、プロジェクト・ファイルのストリーム定義から情報を取得します。 プロジェクト・ファイル名に関する標準はありません。 Event Stream Processor の 2 つのインストール環境でストリーム定義が完全に異なる場合もありますが、すべてのストリーム定義にある、コンポーネントの基本セットは同じものです。

次の手順は、ロイター・マーケットフィード・アダプタを構成するために識別する必要のあるストリーム設定のコンポーネントを示すために、用例のプロジェクトを参照しています。

1. アダプタがデータを提供するプロジェクトを開きます。 ここで示される例では、ロイター・マーケットフィード・アダプタ配布と共に提供される `$ESP_REUTERS_HOME/examples/example.ccl` ファイルです。
2. ソース・ストリームの名前を見つけます。 開始 SourceStream タグは、ストリームの名前を ID 属性の値として指定します。 この例の最初のソース・ストリームの名前は、"stream1" です。

   サブスクリプションのためにロイター・マーケットフィード・アダプタによって使用されるストリームは、常にソース・ストリームである必要があります。
3. キー・フィールドを決定します。 開始 SourceStream タグと終了 SourceStream タグの間の各カラム・エントリで、キー属性が "true" に設定されていることを検証します。 この例では、"stream1" に 1 つのキー・フィールド "symbol" があります。
4. ソース・ストリーム定義のカラム・エントリの数と順序を間違わないように書き留めます。

   入力アダプタ・マップ・ファイルで、同じセットのデータを同じ順序でリストします。

入力アダプタ・マップ・ファイルの作成

アダプタ・マップ・ファイルを作成して、ロイター・マーケットフィード・インプット・アダプタと Event Stream Processor の間のインタフェースを設定します。

次の手順は、RMDS からの更新情報を `example.ccl` ファイルで定義されているソース・ストリームにマップします。 このファイルは、用例のマップ・ファイルと共に `$ESP_REUTERS_HOME/examples` ディレクトリにあります。

1. エディタを使用して新しいマップ・ファイルを開きます。
2. ファイルの先頭行に次を入力して、アダプタ・マップ・ファイルが XML バージョン 1.0 に準拠することを指定します。
   ```
   xml version="1.0" encoding="UTF-8"
   ```
3. 次のドキュメント・タイプ宣言を入力して、これはアダプタ・マップ・ファイルで、このファイルを構成する別のファイルがあることを指定します。
   ```
   <!DOCTYPE adapter [
   <!ENTITY rmdsFields SYSTEM "rmds.sm.mf.xml">
   ]>
   ```
4. 開始 adapter タグと終了 adapter タグを追加します。 開始 adapter タグで、アダプタの名前を指定します。次に例を示します。
   ```
   <adapter name="mySubscribeAdapter1">
   </adapter>
   ```
5. 開始 adapter タグの後に、パブリケーション要素を追加します。 このアダプタのログ・メッセージで使用される名前と、アダプタがデータを Event Stream Processor に配信する方法を規定するために必要な他の属性を指定します。

   次に例を示します。
   ```
   <publication name="RMDS Adapter - low latency" retryInterval="5" /
   >
   ```

   この例には retryInterval 属性もあり、その値は、アダプタに対して、Event Stream Processor への接続に失敗した場合に、5 秒待機してから再試行することを指示しています。
6. publication 要素の後に、開始 streamMaps タグと終了 streamMaps タグを追加します。これらのタグ間には、RMDS FID と Event Stream Processor ストリームのカラムとの間の実際のマッピングを行う streamMap 要素を指定します。 各 streamMap は、1 つのみの Event Stream Processor ストリームにマップされます。
   ```
   <streamMaps>
   </streamMaps>
   ```

   streamMaps セクションには複数の streamMap を指定できるので、アダプタの 1 つのインスタンスは、RMDS データを複数の Event Stream Processor ストリームに提供できます。

7. RMDS データの送信先となる各 Event Stream Processor ストリームの streamMap 要素を入力します。 streamMap ごとに次を実行します。
   a) RMDS データの送信先となる Event Stream Processor ストリームの名前を name 属性の値として指定する、開始 streamMap タグを入力します。
   b) 終了 streamMap タグを入力します。
   c) streamMap タグ間に、ターゲット・ストリーム定義で定義されているカラムごとに 1 つのマッピング要素を追加します。 これは、マップ・ファイル自体内で行うことも、マップ・ファイル内でエンティティとして指定される別のファイル内で行うこともできます。
   ```
   <streamMap name="stream1" flags="NO_SHINE">
   &rmdsFields;
   </streamMap>
   ```
8. streamMaps セクションの後に、以下の rfa 要素を追加します。
   - Reuters 設定ファイルの絶対パスとファイル名を指定する config 属性。
   - Reuters 設定ファイルで使用されるセッション名に対応するセッション名を指定する sessionName 属性。

   ```
   <rfa config="$ESP_REUTERS_HOME/config/rfasub.cfg"
   sessionName="Session1" />
   ```

   rfa 要素には、アダプタがブランクを処理する方法を変更する属性を指定することもできます (デフォルトでは、ブランクはゼロに変換されます)。 blank 属性の値を指定することも、blankInt32、blankInt64、blankMoney、blankString、blankDate、blankTimestamp の各属性を使用して直接に各データ型の値を指定することもできます。 データとして想定されるいずれの値とも競合しない値を指定する必要があります。 入力アダプタと出力アダプタの両方を使用する場合、両方のアダプタの各属性に同じ値が指定されていることを確認します。
9. rfa 要素と終了 adapter タグとの間に、開始 itemLists タグと終了 itemLists タグを追加します。開始 itemLists タグの入力時には、次を行います。
   - アダプタが受信する RMDS データの送信元となる Reuters サービスを service 属性の値として指定する。
   - RMDS データを受信する Event Stream Processor ストリームの名前を、stream 属性の値として指定する。

   ```
   <itemLists service="IDN_RDF" stream="stream1">
   </itemLists>
   ```

   itemLists タグには、開始 itemList タグと終了 itemList タグのペアを 1 つ以上指定できます。
10. itemLists タグ間に、アダプタがサブスクライブする RIC の各個別リストの開始 itemList タグと終了 itemList タグを追加します。
11. itemList タグ間に、リストに追加する各 RIC の item 要素を追加します。 item 要素の入力時には、次を行います。

a) アダプタがサブスクライブする RIC を、name 属性の値として指定します。
b) (オプション) 使用するキューの名前を rfaQueue 属性の値として指定します。 rfaQueue を指定すると、キューを処理する個別のスレッドが生成されます。
c) (オプション) 使用するサービスの名前を指定します。

次に例を示します。

```
<itemList>
<item name="AAPL.O" rfaQueue="queue1" />
<item name="CSCO.O" />
</itemList>
```

入力アダプタの実行
ロイター・マーケットフィード・インプット・アダプタの設定が完了したら、実行します。

前提条件
アダプタが設定されていること。

手順

1. **esp_server** が実行しており、プロジェクトがロードされて起動されていることを確認します。
2. **esp_rmds** コマンドを使用してアダプタを起動します。
   a) Event Stream Processor が認証を必要とせずに実行している場合、次のコマンドを使用してアダプタを起動します。

   ```
   esp_rmds -a in -f mapfile -p cluster_host:cluster_port/
   workspace/project
   ```

   このコマンドでは、アダプタの接続先のプロジェクトの該当する mapfile、cluster_host、cluster_port、作業領域、プロジェクト名を使用します。
   b) Event Stream Processor がいずれかの認証を使用して実行している場合、アダプタを起動するためのコマンドで必要な追加引数についての情報を入手するために、「コマンドの使用方法」を参照してください。

   コマンドの実際の使用方法は、Event Stream Processor の起動方法によって異なります。 アダプタは、互換性のあるオプションを使用して起動する必要があります。 例示されているコマンド文字列には、暗号化も認証も指定されていません。いずれか、または両方を指定できます。
3. アダプタは、最初に Event Stream Processor に接続し、次に RMDS に接続することによってサブスクリプションを起動します。 両方の接続が、すべてのデータ・フローに関して動作している必要があります。

アダプタのログの出力先を stderr にする場合、ここに示されているように、stdout と stderr をログ・ファイルにリダイレクトすることをおすすめします (たとえば、上記のコマンド・ラインに `>& myrmdslog &` を追加します)。

アダプタのテスト

アダプタが期待どおりに動作しない場合は、**esp_rmds** コマンドを実行し、アダプタがロイター・マーケット・データを Event Stream Processor に送信していることを検証して、簡単なサニティ・チェックを行えます。

- **esp_rmds** を実行。
  ```
  esp_rmds -v
  ```
  このコマンドは、バージョン情報を返します。 接続先の Event Stream Processor が使用するアダプタのバージョンと互換性があることを確認します。
- ロイター・マーケットフィード・アダプタがロイター・マーケット・データを Event Stream Processor に送信しているかどうかを簡単に確認するために、3 つの方法が用意されています。
  - スタジオまたは **esp_subscribe** コマンドを使用して、Reuters データを受信するように設定されているストリームの出力をチェック。
  - リダイレクトされたアダプタのログ・ファイル (アダプタ・マップ・ファイルで指定されます) または Reuters サブスクライバ・ログ (設定ファイル `rfasub.cfg` で指定されます) に対して tail コマンドを使用して、アクティビティをチェック。
  - -d7 オプションを指定して **esp_rmds** コマンドを実行し、詳細な出力を生成。

複数の RIC

入力アダプタを設定する場合、通常、複数の RIC を指定します。

これを行うには、いくつかの方法があります。

- 名前をマップ・ファイルに直接入力するか、XML ENTITY インクルード・ファイルを使用することによって、個々の RIC を指定。
- Reuters からのチェーン RIC を指定。
- Event Stream Processor を使用して RIC のリストを指定する動的ウォッチ・リストを作成。
- 上記のオプションを組み合わせて使用。

個々の RIC

マップ・ファイルの itemList セクションに追加する各 RIC の項目要素宣言を入力します。

この例を以下に示します。

```
<itemLists service="SSL_PUB" stream="stream1">
<itemList>
<item name="CSCO.O"/>
<item name="K.N"/>
<item name="KBN.N"/>
<item name="KBR.N"/>
<item name="ACAM.ARC"/>
<item name="IBM.ARC"/>
</itemList>
</itemLists>
```

この方法では、RIC のリストが非常に大きくなったり、リストの変更が頻繁に発生したりする場合に、リストを作成したり、保守したりすることが困難になる可能性があります。 たとえば、NYSE で取引される株式のすべての場合です。 同じストリームのすべての RIC が、同じ FID セットを使用する必要があります。 FID は取引所ごとに異なることがあるので、取引所ごとに異なる itemList と streamMap を使用します。

チェーン RIC

チェーン RIC の名前を指定すると、Reuters はそれを個々の RIC のリストに変換します。 チェーン RIC は、通常、単一のマーケットからのすべての RIC で構成されるか、S&P 500 または Russell 2000 などの単一の指数のすべての RIC で構成されます。

たとえば、ダウ・ジョーンズ指数と SIAC エンティティのチェーン RIC を指定するには、chainMap セクションを追加します。

```
<streamMap name="chainMap" chain="1" >
<itemName /> <dataField name="REF_COUNT" />
<dataField name="NEXT_LR" /> <dataField name="PREF_LINK" />
<dataField name="LINK_1" /> <dataField name="LINK_2" />
<dataField name="LINK_3" /> <dataField name="LINK_4" />
<dataField name="LINK_5" /> <dataField name="LINK_6" />
<dataField name="LINK_7" /> <dataField name="LINK_8" />
<dataField name="LINK_9" /> <dataField name="LINK_10" />
<dataField name="LINK_11" /> <dataField name="LINK_12" />
<dataField name="LINK_13" /> <dataField name="LINK_14" />
<dataField name="LONGNEXTLR" /> <dataField name="LONGPREVLR" />
<dataField name="LONGLINK1" /> <dataField name="LONGLINK2" />
<dataField name="LONGLINK3" /> <dataField name="LONGLINK4" />
<dataField name="LONGLINK5" /> <dataField name="LONGLINK6" />
<dataField name="LONGLINK7" /> <dataField name="LONGLINK8" />
<dataField name="LONGLINK9" /> <dataField name="LONGLINK10" />
<dataField name="LONGLINK11" /> <dataField name="LONGLINK12" />
<dataField name="LONGLINK13" /> <dataField name="LONGLINK14" />
</streamMap>
```

次に、それらの名前を itemList セクションに入力します。

```
<itemList stream="stream1" service="IDN_RDF" >
<item name="0#.DJI" /> <!-- The Dow Jones Index -->
```

```
<item name="0#SIAC" /> <!-- The entities of SIAC -->
</itemList>
```

チェーンの詳細については、`$ESP_REUTERS_HOME/examples` ディレクトリの `chain.example.map.xml` ファイルにある例を参照してください。 Reuters チェーン RIC の詳細については、Reuters から提供されている、選択した取引所向けの『Reuters Venue Guide』を参照してください。

動的ウォッチ・リストの作成
動的ウォッチ・リストの作成手順は、多少複雑ですが、柔軟に行うこともできます。 チェーン RIC は Reuters によって定義されているものに制限されていますが、この手法を使用すると、独自にカスタマイズされた RIC のリストを指定できます。

前提条件
データを受信するための定義済みソース・ストリーム (MyInfoStream) と、手動で編集した RIC のリストが必要です。

手順

この手法も動的です。 以下の手順を使用して設定されたストリーム上で挿入または削除が発生すると、該当する RIC への RMDS サブスクリプションが起動するか、停止します。

1. サブスクライブ先の RIC のリストをアダプタにパブリッシュする、Event Stream Processor 上のストリーム (たとえば、MyListStream) を定義します。 このストリームでは、次のカラムが必要です。

カラム	説明
symbol	アダプタがサブスクライブする RIC 記号チッカ (たとえば、CSCO.O) を指定。
service	その RIC のデータを取得するためにサブスクライブする RMDS サービスを指定。
stream	アダプタがこの RIC のデータをパブリッシュするストリーム (たとえば、MyInfoStream) の名前を指定。

   4 つ目のカラム rfaQueue を指定することもできます (オプション)。
2. 最初のストリームによって要求されるデータを受信する、Event Stream Processor 上の 2 つ目のストリーム (たとえば、MyInfoStream) を定義します。
3. マップ・ファイルを編集して、サブスクリプションを追加します。

```
<subscriptions>
<subscription name="subscription1" flags="BASE" >
```

```
<stream name="MyListStream" >
<name column="3" /> <!-- symbol -->
<field column="1" name="service"/>
<field column="2" name="stream"/>
</stream>
</subscription>
</subscriptions>
```

4. 必要とする RIC のセットを指定し、それらにサブスクライブするために作成した最初のストリーム (たとえば、MyListStream) にそれらを送信します。
   a) ストリームが期待するのと同じ 6 つのカラムをカンマ区切り値 (CSV) フォーマットで持つファイルを作成します。カラムは以下のとおりです。 受信しているデータの送信元ストリーム、opcode、サービス、証券コード、データの送信先ストリーム。

      たとえば、エディタを使用して新しいファイル (`RIClist.csv`) を開き、これらの行を挿入します。

      ```
      MyListStream,p,,IDN_RDF,MyInfoStream,CSCO.O
      MyListStream,p,,IDN_RDF,MyInfoStream,K.N
      MyListStream,p,,IDN_RDF,MyInfoStream,KBN.N
      MyListStream,p,,IDN_RDF,MyInfoStream,KBN.R
      MyListStream,p,,IDN_RDF,MyInfoStream,ACAM.ARC
      MyListStream,p,,IDN_RDF,MyInfoStream,IBM.ARC
      ```

   b) **esp_convert** コマンドと **esp_upload** コマンドを使用してこれらのファイルからデータを Event Stream Processor に送信します。次の例は、すべての Sybase コマンド・ライン・ツールがデフォルトのディレクトリにインストールされており、それらのディレクトリが PATH 変数に追加されていると想定しています。 そうでない場合は、この例で示されている各コマンドの前に、該当するパスを付加してください。

      たとえば、ローカル・サーバのポート 19011 上のワークスペース ws1 でプロジェクト p1 を実行している Event Stream Processor に、上記の手順で作成したファイルを送信するには、次のコマンドを入力します。

      ```
      cat RIClist.csv | esp_convert -c user:password -d "," -p
      localhost:19011/ws1/p1 | esp_upload -c user:password -p
      localhost:19011/ws1/p1
      ```

   c) アダプタを起動します。

      ```
      esp_rmds -f mapfile -d7 -c user:password -p localhost:19011/
      ws1/p1 >& logfile &
      ```

      アダプタと Event Stream Processor が別のマシンに存在する場合は、上記のコマンドで -p の後に、localhost の代わりにリモート・ホストの名前を入力します。

出力アダプタ設定

RMDS をメッセージ・インフラストラクチャとして使用して Event Stream Processor から RMDS にデータをプッシュするように出力アダプタを設定します。

提供するデータとシステムをセットアップする方法を決定してから、出力アダプタを設定します。

受信データの送信元の Event Stream Processor インスタンスについて、以下の情報を取得する必要があります。

- クラスタ環境で適用可能なセキュリティ・オプションと、ワークスペースとプロジェクトの名前。
- 使用される認証メカニズム (Kerberos、RSA、LDAP、または none)。

データの決定

Event Stream Processor に流入するどのカラムからデータをパブリッシュするかを識別します。

ロイター・マーケットフィード・アダプタは、ストリームからのカラムを任意の順序に並び替えることができます。 その出力は定数を構成でき、パブリッシュされた出力は複数のストリームからの値も構成できます。

ロイター・マーケットフィード・アウトプット・アダプタの出力を計画する場合、次の点に注意する必要があります。

- データのパブリッシュ元の各ストリームに対して、出力アダプタ・マップ・ファイルに一意なキーを指定できる必要がある。 このアダプタはデータを RMDS に送信するので、一意なキーは RIC である必要があります。
- いずれかのストリームからパブリッシュされる各データ・カラムは、一意な FID にマップされる必要がある。
- 1 つのカラムからのデータは、パブリッシュされる出力内で繰り返し出現できる。これによって、日時値を個別の日付と時刻の値としてパブリッシュできます。
- 作業しているストリームが複数のサービスから同じ FID についてのデータを受信する場合、これらのデータ項目をサービスごとに区別し、データをサービスごとに個別に転送するようにアダプタを設定できる。
- ロイター・マーケットフィード・アダプタは RMDS に初めてパブリッシュするときに、設定されているすべてのカラムの値をパブリッシュする。その初期イメージの後、これらのカラムの更新が発生するごとに、アダプタは個々のカラムの更新情報のみをパブリッシュします。

管理上の決定
プロジェクトに関して、いくつかの管理上の決定を行う必要があります。

決定事項	説明
セッション名	プロジェクトとアダプタ・マップ・ファイルを結び付ける任意の文字列。 わかりやすい文字列を使用します。
ロギングとストリーム出力用のディレクトリ	アダプタは、独自のログ・メッセージを書き込み、Reuters ログ・メッセージの個別のセットを生成可能。 設定では、これらのログ・ファイルが書き込まれるかどうかと、どこに書き込まれるかを指定します。
Sybase ユーザ・アカウント	Event Stream Processor の起動時に認証を指定していない場合を除き、アダプタが使用する、有効な Event Stream Processor ユーザ・アカウントを指定。

Reuters 情報
ロイター・マーケットフィード・アダプタが RMDS にパブリッシュできるようにするには、Reuters から提供されているいくつかの情報を使用する必要があります。

- アダプタがデータを転送する Reuters サービスの名前
- RMDS によって使用される、有効な RIC (Reuters Instrument Code) と Field ID (FID) の最新のリスト
- Reuters によって割り当てられる Product Permission Code

アダプタは Reuters データ・アクセス制御システム (DACS) をサポートしていないので、Product Permission Code が、RMDS 上で転送する情報へのアクセスを許可するためのメカニズムを提供するために必要です。

FID のリスト $ESP_REUTERS_HOME/config/appendix_a が、Reuters アダプタ配布の一部として提供されています。 最新のリストと他の情報については、Reuters のテクニカル・サポートから取得してください。

Event Stream Processor のカラムのデータ型は、それをフィードするロイター FID のデータ型と互換性がある必要があります。 次の表は、一致可能な Event Stream Processor と FID のデータ型の組み合わせを示します。

Event Stream Processor のデータ型	Reuters のデータ型
`integer`, `long`	`integer` または `price`
`money`, `float`	`price`

Event Stream Processor のデータ型	Reuters のデータ型
`string`	`alphanumeric`
`date, timestamp`	`date` または `time`

プロジェクトからストリーム情報の取得
プロジェクトから必要な情報を収集します。

出力アダプタを設定する最初の手順では、Event Stream Processor に流入するどのデータ要素がパブリッシュされるかを決定します。 Reuters アダプタを介して RMDS 上でパブリッシュされる項目で構成されるプロジェクトを選択 (または定義) したら、RMDS に送信するデータの取得元となるストリームから情報を収集します。

各ストリーム定義は、Event Stream Processor の起動時にインスタンス化されるデータ・ストリームを指定します。 ストリーム定義は、以下の機能を提供します。

- ストリームの一意な ID を指定する。
- データ・ストリームの各ローの一意なキーとして使用されるカラムを識別する。

Reuters アダプタによって RMDS に送信される情報を提供するストリームを決定したら、プロジェクト・ファイルのストリーム定義から情報を取得します。 プロジェクト・ファイル名に関する標準はありません。 Event Stream Processor の 2 つのインストール環境でストリーム定義が完全に異なる場合もありますが、すべてのストリーム定義にある、コンポーネントの基本セットは同じものです。

1. アダプタがデータを提供するプロジェクトを開きます。 ロイター・マーケットフィード・アダプタ配布には、サンプル・プロジェクト `$ESP_REUTERS_HOME/examples/example.ccl` があります。
2. プロジェクトで定義されている各ストリームの定義を使用して、以下を実行します。
   a) そのストリームの開始タグの ID 属性からストリームの名前を取得します。
   b) RIC を含むカラムに対してキー属性が "true" に設定されていることを検証し、そのカラムを書き留めます。 この例では、"stream1" と "orderbookStream" の両方が、キー・フィールドとして識別される "symbol" という名前のカラムに RIC を持ちます。
   c) アダプタが RMDS に送信するデータを決定します (存在する場合)。
3. RMDS に送信するデータで構成されるストリームと、ストリーム定義のどこにそれがあるかを、間違わないように書き留めます。

出力アダプタ・マップ・ファイルで、パブリッシュする各カラムを参照します。

出力アダプタ・マップ・ファイルの作成
アダプタ・マップ・ファイルを作成して、出力アダプタと Event Stream Processor の間のインタフェースを設定します。

次の例は、RMDS からの更新情報を `example.project.xml` ファイルで定義されているソース・ストリームにマップします。

1. エディタを使用して新しいマップ・ファイルを開きます。
2. ファイルの先頭行に次を入力して、アダプタ・マップ・ファイルが XML バージョン 1.0 に準拠することを指定します。
   ```
   xml version="1.0" encoding="UTF-8"
   ```
3. 開始 adapter タグと終了 adapter タグを追加します。
   ```
   <adapter>
   </adapter>
   ```
4. 次の属性の指定された rfa タグを追加して、RMDS へのアダプタのインタフェースの設定を定義します。

属性	説明
config	Reuters 設定ファイルのフル・パス名を指定。
fidFile	有効な FID をすべてリストする、Reuters 提供のファイルのフル・パス名を指定。
enumFile	各列挙型とその値の範囲をリストする、Reuters 提供のファイルのフル・パス名を指定。
serviceName	アダプタがデータを RMDS に送信できるようにするために Reuters によって提供されているサービスの名前を指定。
sessionName	Reuters 設定ファイル `rfasub.cfg` にある sessionName 値を指定。

   たとえば、アダプタ配布に同梱されているファイルを使用します。
   ```
   <rfa config="$ESP_REUTERS_HOME/config/rfapub.cfg"
   fidFile="$ESP_REUTERS_HOME/config/appendix_a"
   enumFile="$ESP_REUTERS_HOME/config/enumtype.def"
   serviceName="IDN_RDF" sessionName="Session1" />
   ```
5. rfa 要素と終了 adapter タグとの間に、開始 subscriptions タグと終了 subscriptions タグを追加します。
   ```
   <subscriptions>
   </subscriptions>
   ```

アダプタは Event Stream Processor にサブスクライブして、RMDS にパブリッシュするデータを取得します。

6. 開始 subscriptions タグと終了 subscriptions タグの間に、開始 subscription タグと終了 subscription タグを追加してサブスクリプションを定義します。 開始 subscription タグに次の属性を指定します。

属性	説明
`name`	このサブスクリプションの一意な名前を指定。
`flags`	このパラメータを "BASE" に設定して、初期値の完全なセットを取得します。 これは、変更されていない値が大量にあるリカバリなどの状況では、好ましくないことがあります。これは、それらの値を取得することによって、他の値の取得に遅延が生じる可能性があるためです。 このような状況では、このパラメータを "NO_BASE" に設定します。

```
<subscription name="subscription1" flags="BASE" >
</subscription>
```

出力アダプタ・マップ・ファイルで定義されている各サブスクリプションは、1 つ以上の Event Stream Processor ストリームを参照する必要があります。

7. ストリーム定義をサブスクリプションに追加します。
   a) 終了 subscription タグの直前に、開始 stream タグと終了 stream タグを挿入します。 開始 stream タグに、ストリームの名前に設定された name 属性を指定します。
   b) Event Stream Processor ストリームのカラムではなく "constant" を使用して Reuters の Permission Code を指定するために、終了 stream タグの直前に constant タグを挿入し、次の属性を指定します。

属性	説明
`name`	Reuters FID の "PROD_PERM" を指定。
`value`	RMDS にパブリッシュする許可を証明する、Reuters によって発行された Permission Code を指定。

   c) 開始 stream タグの直後に name タグを挿入し、プロジェクトの証券コードまたは RIC を持つカラムの前のカラムに属性カラムを設定します。 たとえば、証券コードまたは RIC を持つカラムがプロジェクトの最初のカラムの場合、カラムの値を "0" に設定します。
   d) 開始 name タグの直後に、stale タグを挿入し、属性カラムをプロジェクトの値の位置よりも 1 少ない値に設定します。
   e) stale タグと constant タグの間に、RMDS に送信するストリームのデータ・カラムごとに field タグを追加し、 次の属性を指定します。

属性	説明
column	このパラメータをカラムの名前、または数値で表される位置 (プロジェクト内の値の位置よりも 1 少ない数) に設定。
name	このデータの Reuters FID を指定。

浮動小数点数のデータ型のフィールドに対しては、precision 属性も指定できます。この属性は、RMDS に送信される値の小数点以下の桁数を設定します。 次に例を示します。

```
<stream name="stream1" >
<name column="0"/>
<stale column="3" />
<field column="4" name="BID" precision="5" />
<field column="5" name="ASK" precision="0" />
<field column="6" name="TRDPRC_1"/>
<field column="7" name="ACVOL_1"/>
<constant name="PROD_PERM" value="1"/>
</stream>
```

出力アダプタの実行
アダプタの設定が完了したら、実行します。

前提条件
アダプタが設定されていること。

手順

1. **esp_server** が実行しており、プロジェクトがロードされて起動されていることを確認します。
2. **esp_rmds** コマンドを使用してアダプタを起動します。
   a) Event Stream Processor が認証または暗号化を必要とせずに実行している場合、次のコマンドを使用してアダプタを起動します。
   ```
   esp_rmds -a out -f mapfile -p cluster_host:cluster_port/
   workspace/project
   ```
   このコマンドでは、アダプタの接続先のプロジェクトの該当する mapfile、cluster_host、cluster_port、作業領域、プロジェクト名を使用します。
   b) Event Stream Processor が暗号化またはいずれかの認証を使用して実行している場合、「コマンドの使用方法」を参照して、アダプタを起動するためのコマンドで必要な追加引数についての情報を入手してください。

コマンドの実際の使用方法は、Event Stream Processor の起動方法によって異なります。 アダプタは、互換性のあるオプションを使用して起動する必要があ

ります。 例示されているコマンド文字列には、暗号化も認証も指定されていません。いずれか、または両方を指定できます。

3. アダプタは、最初に Event Stream Processor に接続し、次に RMDS に接続することによってサブスクリプションを起動します。 両方の接続が、すべてのデータ・フローに関して動作している必要があります。
   アダプタのログの出力先を stderr にする場合、ここに示されているように、stderr をログ・ファイルにリダイレクトすることをおすすめします (たとえば、上記のコマンド・ラインに `>& myrmdslog &` を追加します)。

アダプタのテスト

アダプタが期待どおりに動作しない場合は、**esp_rmds** コマンドを実行し、アダプタがロイター・マーケット・データを Event Stream Processor に送信していることを検証して、簡単なサニティ・チェックを行えます。

- **esp_rmds** を実行。
  ```
  esp_rmds -v
  ```
- このコマンドは、アダプタのリリース番号とソース・ツリーのリビジョン番号をアンダースコア文字で区切って返す。 使用しているアダプタと Event Stream Processor のバージョンに互換性があることを確認します。
- ロイター・マーケットフィード・アダプタが RMDS にパブリッシュしていることを確認する方法が、いくつか用意されている。
  - コンソール出力のリダイレクト先のアダプタ・ログ・ファイル、または任意の Reuters パブリッシャのログ・ファイル (`rfapub.cfg` で指定されています) に対して tail コマンドを使用して、アクティビティを検索する。
  - **esp_subscribe** コマンドを使用してアウトバウンド・ストリームを調査し、値が変化していることを検証する。
  - RMDS ツールを使用して、出力アダプタによって提供されている RIC にサブスクライブする。
  - 入力アダプタを使用して、出力アダプタにサブスクライブする。

従属マップ・ファイルの作成

従属マップ・ファイルを作成して、マップ・ファイル設定の一部を保持します。

入力アダプタ・マップ・ファイルまたは出力アダプタ・マップ・ファイルの一部を別のファイルに書き込むことによってもパフォーマンスが向上します。たとえば、サブスクリプション設定をマップ・ファイルに維持し、アダプタがサブスクライブする RIC のリストを別ファイルで維持します。

1. マップ・ファイルが存在するディレクトリに移動します。
2. 拡張子 `.xml` を持つ新しいファイルを作成します。

XML バージョンの宣言を追加する必要はありません。

3. 選択した内容をマップ・ファイルから新しいファイルに挿入します。
   追加する内容は、個別に格納すると決定した、マップ・ファイルの部分により異なります。
4. (オプション) コメントを新しいファイルに追加します。
5. 編集が完了したら、ファイルを保存します。

メイン・マップ・ファイルの変更

従属ファイルを参照するように、メイン・マップ・ファイルを変更します。

1. メイン・マップ・ファイルの先頭行が以下であることを確認します。
   ```
   <?xml version="1.0"?>
   ```
2. 次の行を、XML バージョンの宣言と開始 adapter タグの間に追加します。
   ```
   <!DOCTYPE adapter SYSTEM "adapter.dtd" [
   ]>
   ```
3. 各従属マップ・ファイルについて、以下を実行します。
   a) 追加したばかりの 2 行の間に以下のようなエントリを追加します。
      ```
      <!ENTITY SUBREF SYSTEM "SUBFILE">
      ```
      ここで、SUBREF は従属ファイルを参照するために使用される文字列で、SUBFILE は従属ファイル自体のパスとファイル名です。 ファイル名とパスを引用符で囲みます。
   b) 従属マップ・ファイルに挿入した内容を削除します。
   c) 以下のような文字列を挿入して、従属マップ・ファイルからの内容をインクルードします。
      ```
      &SUBREF;
      ```
      ここで、SUBREF は従属ファイルを参照するように指定した文字列です。

例

入力アダプタをマップ・ファイル (`subInclude.map.xml`) で設定して、2 つの従属ファイル (`RIClist1.sm.mf.xml` と `RIClist2.sm.mf.xml`) を参照します。

マップ・ファイル `subInclude.map.xml` は、入力アダプタを設定して、2 つの従属ファイルを参照します。各従属ファイルには、アダプタがサブスクライブする RIC のリストがあります。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE adapter [
<!ENTITY RIClist1 SYSTEM "RIClist1.sm.mf.xml">
<!ENTITY RIClist2 SYSTEM "RIClist2.sm.mf.xml">
<!ENTITY rmdsFields SYSTEM "rmds.sm.mf.xml">
]>
<adapter>
<publication name="RMDS Adapter" retryInterval="5"
```

```
sendAsTransactions="0" flushInterval="1000"
intraSubscribeDelay="100"/>
<streamMaps>
<streamMap name="stream1">
&rmdsFields;
</streamMap>
</streamMaps>
<rfa config="$ESP_REUTERS_HOME/config/rmdsmf.cfg"
sessionName="Inbound" />
<itemLists>
&RIClist1;
&RIClist2;
</itemLists>
</adapter>
```

最初のファイル `RIClist1.sm.mf.xml` の内容は、次のとおりです。

```
<!-- This fragment is meant to be included in an itemLists section.--
>
<!-- These are FX RICs -->
<itemList service="IDN_RDF" stream="stream1">
<item name="GRMN.O"/>
<item name="INTC.O"/>
<item name="KLAC.O"/>
<item name="XLNX.O"/>
<item name="YHOO.O"/>
</itemList>
```

2 つ目のファイル `RIClist2.sm.mf.xml` の内容は、次のとおりです。

```
<!-- This fragment is meant to be included in an itemLists section.--
>
<!-- These are FX RICs -->
<itemList service="IDN_RDF" stream="stream1">
<item name="AUD="/>
<item name="CAD="/>
<item name="DKKTN="/>
<item name="GBPSW="/>
<item name="GBPTN="/>
<item name="JPYSN="/>
<item name="JPYSW="/>
<item name="JPYTN="/>
<item name="HKD="/>
<item name="SGDSW="/>
<item name="ZAR="/>
<item name="ZARSN="/>
</itemList>
```

パフォーマンス・チューニング

入力アダプタのパフォーマンスを微調整するために使用できる、4 つの属性があります。

属性	説明
`flushInterval`	データの累積時に待機する時間間隔をマイクロ秒単位 (たとえば、5000 マイクロ秒 = 5 ミリ秒) で指定。 この間隔の終了時に、累積されたすべてのイベントが Event Stream Processor に送信されます。 送信イベントの間隔を長くすると、多くのイベントがメッセージに格納され、通信オーバヘッドが削減されます。ゼロ以外の値を flushInterval に指定すると、イベントが時間を基にして累積されます。
`maxRecordsPerBlock`	Event Stream Processor にアダプタが 1 回に送信する、累積されたイベントの最大数を指定。 累積されたイベントの数がこの値よりも大きいと、エンベロープまたはトランザクションが、指定した値以下のイベントで構成されるフラグメントに分割されます。 たとえば、累積されたイベントの数が 1024 (この数では、Event Stream Processor Gateway のインバウンド・キューがすぐにいっぱいになります) を超えることが予想される場合、maxRecordsPerBlock を 500 などの値に設定して、いっぱいになるのを防ぎます。
`pendingLimit`	イベントを累積する数のしきい値を指定。この値になると、累積されたイベントが Event Stream Processor に送信されます。 このパラメータをゼロに設定すると、イベントが発生するとすぐにアダプタはそのイベントをパブリッシュします (遅延が最も小さくなります)。ただし、ネットワークのオーバヘッドが非常に高くなります (更新情報ごとに 1 つの TCP/IP パケット)。 このパラメータを大きな値に設定すると、アダプタは多くのイベントが累積されるまで待機し、TCP/IP パケットに効率よく格納して、Event Stream Processor に送信します。 これによって、通信処理の負荷は削減されますが、アダプタと Event Stream Processor の両方で遅延が増加します。

属性	説明
sendAsTransactions	このパラメータは、イベントがエンベロープまたはトランザクションのいずれで送信されるかを制御。 このパラメータは、ストリームごとに指定できます。 このパラメータを true に設定すると、Event Stream Processor は、イベントのグループを 1 つのトランザクションとして扱いします。 トランザクションは、通常、アプリケーションレベルのワークロード削減をもたらします。これは、Event Stream Processor が、1 つのトランザクション内で同じ値 (同一のキー・カラムによって判断されます) に対する複数のイベントを 1 つのイベントにまとめるためです。 また、トランザクションに削除がある場合、削除の前の更新が廃棄されるので、さらなるワークロード削減が達成されます。 このパラメータを false に設定し、低遅延モード (pendingLimit と flushInterval の両方をゼロに設定) ではない場合、maxRecordsPerBlock を使用してエンベロープのサイズを制御します。 上記で説明されているように、通信オーバヘッドが削減されますが、トランザクション上の削減は得られません。 これは、マーケット・データ整合アプリケーションなど、各イベントをすべて必要とするアプリケーションで推奨される設定です。 一般的なルールとして、最新の更新情報のみが重要な相場値ベースのアプリケーションでは、トランザクションの設定がはるかに効率的です。 総量を計算するために各イベントをすべて処理する必要がある取引では、エンベロープを使用する必要があります。

flushInterval と pendingLimit の両方が使用されている場合、すべてのイベントが flushInterval の時間を待たずに送信され、pendingLimit (またはそれ以上) のイベントが到着すると、それらはすぐに送信されます。 アダプタは、flushInterval で指定された時間が経過するまで待機し、イベントが累積された場合には、それらを送信します。 アダプタが以前のイベントを送信しているときに pendingLimit の値以上のイベントが累積された場合、アダプタは新しいイベントをすぐに送信します (flushInterval の時間が経過するまで待機することはありません)。 アダプタがイベントを送信しているときに pendingLimit の値よりも少ないイベントが累積されている場合、アダプタは flushInterval の時間が経過するまで待機します。

rfaQueue 属性も itemLists、itemList、または item の要素のレベルで使用できます。この属性を指定すると、名前付き rfaQueue 上で Reuters から要素がサブスクライブされます。 各 rfaQueue は Reuters アダプタ内の独自のスレッドによって処理されま

す。 要求を複数のスレッドに分散することで、CPU の使用率を向上させ、遅延を削減し、全体的なアダプタのスループットを改善できます。

同じ RIC のすべてのイベント (イメージと更新) は同じキュー上で Reuters から到着するので、到着の順序の整合性は、個々の RIC すべてで維持されます。 要素のいずれにも rfaQueue を指定しない場合、すべての RIC に対して 1 つのデフォルト・キュー ("default") が使用されます。

コマンドの使用

ロイター・マーケットフィード・アダプタは、Reuters Market Data System (RMDS) からのデータを Event Stream Processor のデータに、およびその逆方向に変換します。

シノプシス

```
esp_rmds -f mapFile -p host:port/workspace/project [ OPTION ...]
```

説明

esp_rmds は、入力アダプタまたは出力アダプタのいずれかとして動作できます。 入力アダプタは、RMDS からのデータを Event Stream Processor に渡します。 出力アダプタは、Event Stream Processor からのデータを RMDS に渡します。 1 つのアダプタ・インスタンスは、両方向を処理できません。 入力アダプタと出力アダプタを必要とする場合、2 つの個別のアダプタ・インスタンスが必要です。

接続を記述するメタデータには、マップ・ファイル、設定ファイルなどのいくつかのパートがあります。また、Event Stream Processor の実行中のインスタンスに常駐する設定ストリームのパートがあることがあります。

限定された Level 2 データのみが、RMDS マーケットフィードを介して利用できます。 完全なオーダーブック深度に対しては、ロイター OMM アダプタ (**esp_rmdsomm**) を使用します。

プロセスは、デーモンとして実行し、設定をマップ・ファイルから取得します。 SIGHUP を処理するので、`kill -s SIGHUP pid` (Linux の場合) または `kill -s HUP pid` (Solaris の場合) を入力して、アダプタをシャットダウンできます。ここで、pid は **esp_rmds** デーモンのプロセス ID で、ps コマンドを使用して取得できます。 HUP シグナルではなく KILL シグナルを使用することによって、システム・リソースの完全なクリーン・アップを防止できます。

アダプタがインストールされているディレクトリの下には、doc、examples、config の 3 つのディレクトリがあり、追加情報が格納されています。 doc ディレクトリには、さまざまな設定オプションを記述する Reuters README ファイルがあります。 examples ディレクトリには、多くの機能を示す、いくつかの用例マップ・ファイルがあります。 config ディレクトリには、RMDS 設定ファイルの例が

あります。 少なくとも RMDS 設定ファイルを、サイト固有の情報を使用して変更する必要があります。 一般的に、マップ・ファイルも Event Stream Processor に一致するように変更します。

必須の引数

- **-f mapFile –** マーケット・データと RMDS との間をマップするために必要なメタデータが構成されているマップ・ファイルを指定。
- **-p host:port/workspace/project –** サーバ (クラスタ・マネージャ) に接続するための URI を指定。 たとえば、`-p localhost:19011/default/prj1` は、コマンドを入力したマシンのポート 19011 を使用する ESP クラスタ・サーバのデフォルトのワークスペースに存在する prj1 と呼ばれるプロジェクトを指定します。

オプション

- **-a in|out –** RMDS マーケットフィード・アダプタのインスタンスがデータを Event Stream Processor に渡しているか、それから渡されたデータを受信しているかを指定。 指定できる値は、`in` と `out` です。 デフォルト値は `in` なので、マーケット・データにサブスクライブするときには、このオプションは、通常、省略されます。

  下位互換性のため、"subscribe" (in) と "publish" (out) は依然として許可されますが、非推奨です。
- **-c user:password –** 認証方法を使用しており、クレデンシャル (Kerberos、PAM、または RSA) を必要とする場合、このオプションはそれらの認証クレデンシャルを Event Stream Processor に渡します。 Event Stream Processor がこれらのクレデンシャルを正常に認証すると、接続が維持されます。それ以外の場合、Event Stream Processor は接続をすぐに閉じます。
- **-d debugLevel –** デバッグ・レベルを設定。有効な範囲は 0 ～ 7 です。0 が最小で、7 が詳細です。 デフォルトは、4 に設定されます。
- **-e –** Event Stream Processor とのすべての接続に、暗号化された OpenSSL ソケットでネゴシエート。このオプションを使用するときには、Event Stream Processor を暗号化モードで起動する必要があります。
- **-F configFile –** RMDS 設定ファイルを指定し、マップ・ファイルで指定されている設定ファイルを上書き。
- **-g gatewayHost –** Event Stream Processor ゲートウェイ・ホストを明示的に指定。
- **-G –** Kerberos 認証を使用。このオプションは、-V gssapi オプションを使用して Event Stream Processor を起動したときに必要です。
- **-h –** このコマンドの構文を説明する簡易なヘルプ・メッセージを表示。
- **-k privateRSAKeyFile –** パスワード認証の代わりに、RSA プライベート・キー・ファイル・メカニズムを使用して認証を実行。 privateRSAKeyFile には、

RSA プライベート・キー・ファイルの絶対パス・ファイル名を指定する必要があります。 このオプションを有効にした場合、-c オプションを使用してユーザ名を指定する必要がありますが、パスワードは必要ありません。 さらに、Event Stream Processor が -k オプションを使用して起動されている必要があります。

- **-l 0|1|2|3 –** ログ・メッセージの送信先を指定。ログ・メッセージを送信しない場合は、`0` を使用します。stderr のみに送信する場合は `1` (デフォルト)、syslog のみに送信する場合は、`2`、stderr と syslog の両方に送信する場合は、`3` を指定します。
- **-r subscribeRetryInterval –** RIC に再サブスクライブするまでの待機時間を秒単位で指定 (デフォルトは、300) (RIC へのサブスクリプションが CLOSED または CLOSEDRECOVER とマークされている場合、データを流れさせるためにその RIC に再サブスクライブする必要があります)。 再サブスクライブの試行を無効にするには、値として `0` を指定します。 定期的に再サブスクライブすることによって、ソースがサブスクライバにとって利用できないなどの一時的な状態から回復できます。 再サブスクライブ試行が失敗するごとに失敗イベントが生成されます。これによって、ステータスが更新されて、項目が失効しているとマークされます。
- **-s streamName –** 検出モードで実行時に使用されるストリームを指定。 このオプションは、コネクタ起動メカニズムによって使用され、マップされたカラムが検出された単一ストリームを指定します。
- **-v –** RMDS マーケットフィード・アダプタのバージョンを印刷して終了。
- **-w retrySeconds –** Event Stream Processor への接続を再試行するまでの待機時間を秒単位で指定。 デフォルトは 5 です。 `0` の指定は、接続を 1 回だけ試行し、その後、再試行しないことを意味します。
- **-x optName –** その他の設定を指定。`-x help` を使用すると、指定可能な値のリストが表示されます。
- **-z publishCount –** 終了するまでに、Event Stream Processor に渡す値の数を指定。デフォルトは 0 で、終了しないことを意味します。
- **-Z subscribeCount –** 終了するまでに、RMDS に渡す値の数を指定。 デフォルトは 0 で、終了しないことを意味します。

例

入力アダプタを起動するには、マップ・ファイル `subexample.map.xml` を使用し、localhost マシンのポート 19011 上の ws1 という名前のワークスペースに存在する project1 という名前のプロジェクトを実行します。

```
esp_rmds -c user:pw -p localhost:19011/ws1/project1 -a in -f
subexample.map.xml
```

環境変数

ロイター・マーケットフィード・アダプタは、環境変数を使用して動作を指定します。

環境変数	使用箇所	説明
ESP_ACCUMULATOR_DELAY	入力	エキスパート： Event Stream Processor への接続を遅延する時間 (秒単位)。
ESP_DISABLE_REPORT_ENCODING_NULL	出力	ブランクから null への変換についての警告を停止 (bool 値で指定。デフォルトは false)。
ESP_FLUSH_INTERVAL	入力	パブリッシュの flushInterval を上書き (マイクロ秒単位)。
ESP_INTRASUBSCRIBE_DELAY	入力	Map 属性を上書き (ミリ秒単位)。
ESP_LOG_CONFIG_EVENTS	両方	設定イベント処理用のログ・レベル (1 ～ 7) を設定 (デフォルトは -1)。
ESP_MARKETFEED_DUMP	出力	生の Reuters メッセージをログにダンプするログ・レベル (0 ～ 7) を設定。
ESP_MAX_RECORDS_PER_BLOCK	入力	パブリッシュの maxRecordsPerBlock を上書き (カウント数)。
ESP_PENDING_LIMIT	入力	パブリッシュの pendingLimit を上書き。
ESP_RETRY_INTERVAL	両方	パブリッシュの retryInterval を上書き。
ESP_REUTERS_HOME	両方	インストール・ディレクトリを指定。

環境変数	使用箇所	説明
ESP_RMDS_DISPATCH	両方	エキスパート：N ミリ秒ごとに RFA をディスパッチ (デフォルトは、10,000)。
ESP_RMDS_EVENT_TRACE	両方	エキスパート：N イベントごとに RFA イベント・トレーシングを有効化 (整数)。
ESP_RMDS_PUBLISH_BUFSIZE	出力	バッファ・サイズを上書き。
ESP_RMDS_PUBLISH_DEBUG_LEVEL	出力	値を表示するには、7 に設定。
ESP_RMDS_PUBLISH_DEBUG_SYMBOLS	出力	記号を表示しないデフォルトの動作が上書きされたときに使用される記号のスペース区切りのリスト。設定されていない場合、すべての記号が使用されます。
ESP_RMDS_SUBSCRIBE_DEBUG_LEVEL	入力	値を表示するには、7 に設定。
ESP_RMDS_SUBSCRIBE_DEBUG_SYMBOLS	入力	上記用の記号のスペース区切りのリスト。 設定されていない場合、すべての記号が使用されます。
ESP_RMDS_SUBSCRIBE_SYMBOL_FORMAT	入力	記号のリスト・フォーマットを指定。 複数行に対しては 0、単一行に対しては 1 を指定します。
ESP_SEND_AS_TRANSACTIONS	入力	Map 属性を上書き。
ESP_SHOW_FIELD_INFO	入力	FID、カラム、spColumn、ストリーム名を表示 (デフォルトは false)。
ESP_SHOW_SP_EVENT_DATA	出力	Event Stream Processor からのイベントのログ・レベル (1 ～ 7) を設定 (デフォルトは、-1)。

入力アダプタ・マップ・ファイルの XML 構文

ロイター・マーケットフィード・インプット・アダプタ用のマップ・ファイルの構文を以下に示します。

```
adapter                    (required, limit one)
  |----publication                    (required, limit one)
  |----streamMaps                     (required, limit one)
  |       '----streamMap              (required)
  |              |----itemName        (required, limit one)
  |              |----serviceName     (optional)
  |              |----sequenceNumber  (optional)
  |              |----itemStale       (optional)
  |              |----dataField       (required)
  |              |----updateNumber    (required)
  |              |----dateTimeField   (optional)
  |              |----FIDListField    (optional)
  |              '----nullField       (optional)
  |----recordTypeMap                  (optional)
  |       '----recordType             (optional)
  |----rfa                            (required, limit one)
  '----itemLists                      (required, limit one)
         '----itemList                (required)
                '----item             (optional)
```

adapter

adapter 要素は、マップ・ファイルのルート要素です。

まとめ

```
adapter                    (required, limit one)
  |----publication                    (required, limit one)
  |----streamMaps                     (required, limit one)
  |       '----streamMap              (required)
  |              |----itemName        (required, limit one)
  |              |----serviceName     (optional)
  |              |----sequenceNumber  (optional)
  |              |----itemStale       (optional)
  |              |----dataField       (required)
  |              |----updateNumber    (required)
  |              |----dateTimeField   (optional)
  |              |----FIDListField    (optional)
  |              '----nullField       (optional)
  |----recordTypeMap                  (optional)
  |       '----recordType             (optional)
  |----rfa                            (required, limit one)
  '----itemLists                      (required, limit one)
         '----itemList                (required)
                '----item             (optional)
```

すべての設定セクションは、**adapter** の開始タグと終了タグの間で構成する必要があります。

親
なし

子
以下の子要素が、アダプタに対して定義されます。 これらの要素すべてが存在している必要があり、指定された順序で配置する必要があります。

名前	稼働条件
publication	1 つだけ必要
streamMaps	1 つだけ必要
recordTypeMap	オプション
rfa	1 つだけ必要
itemLists	1 つだけ必要

属性

名前	説明	稼働条件
name	ログ・エントリでこのアダプタを一意に識別する文字列	オプション

注意
なし

例
adapter 定義で構成される個々の要素については、例を参照してください。

dataField
streamMap 定義で、**dataField** 要素は、ソース・ストリームからの1つのカラムをロイター FID にマップします。

まとめ

```
adapter                    (required, limit one)
  |----publication                    (required, limit one)
  |----streamMaps                     (required, limit one)
  |       '----streamMap              (required)
  |              |----itemName        (required, limit one)
  |              |----serviceName     (optional)
  |              |----sequenceNumber  (optional)
  |              |----itemStale       (optional)
  |              |----dataField       (required)
  |              |----updateNumber    (required)
  |              |----dateTimeField   (optional)
```

```
  |             |----FIDListField    (optional)
  |             '----nullField       (optional)
  |----recordTypeMap                 (optional)
  |       '----recordType            (optional)
  |----rfa                           (required, limit one)
  '----itemLists                     (required, limit one)
         '----itemList               (required)
                '----item            (optional)
```

親
streamMap

子
なし

属性

名前	説明	稼働条件
name	ソース・ストリームのこのカラムに出現するデータ項目を識別するロイター FID	必須
key	true または false のいずれか (このカラムがソース・ストリームの一意なキーの一部であるかどうかによって決定)	「注意」を参照

注意
入力アダプタ・マップ・ファイルの **streamMap** セクションの各要素は、ターゲット・ソース・ストリームの RowDefinition の 1 つのカラムを表している必要があります (streamMap 要素の順序は、RowDef のカラムの順序と一致している必要があります)。 RowDef のカラムがデータ項目 (Bid、Ask など) の場合、対応する **streamMap** エントリは、name 属性が特定の FID を識別する **dataField** 要素である必要があります。 その FID が設定されている更新を RMDS がパブリッシュするごとに、アダプタは、それを対応するローの値として Event Stream Processor のソース・ストリームに送信します。

値を true に設定するには、**key** 属性を使用する必要があります。 このカラムがストリームのキーの一部ではない場合、この属性は省略できます。

例

```
<streamMap name="stream1">
        <itemName key="true"/>
        <FIDListField />
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
```

```
        <dataField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

上記の例は、stream1 のカラム 5 ～ 8 をロイター FID の BID、ASK、TRDPRC_1、ACVOL_1 にマップします。

dateTimeField

streamMap 定義で、**dateTimeField** 要素は、Reuters の日付または時刻の FID (またはそれぞれの 1 つ) をストリームの日付またはタイムスタンプにマップします。

まとめ

```
adapter                    (required, limit one)
  |----publication                    (required, limit one)
  |----streamMaps                     (required, limit one)
  |       '----streamMap              (required)
  |              |----itemName        (required, limit one)
  |              |----serviceName     (optional)
  |              |----sequenceNumber  (optional)
  |              |----itemStale       (optional)
  |              |----dataField       (required)
  |              |----updateNumber    (required)
  |              |----dateTimeField   (optional)
  |              |----FIDListField    (optional)
  |              '----nullField       (optional)
  |----recordTypeMap                  (optional)
  |       '----recordType             (optional)
  |----rfa                            (required, limit one)
  '----itemLists                      (required, limit one)
         '----itemList                (required)
                '----item             (optional)
```

親

streamMap

子

なし

属性

名前	説明	稼働条件
dateName	RMDS によって提供される日付値の FID	「注意」を参照
timeName	RMDS によって提供される時刻値の FID	「注意」を参照

注意
Event Stream Processor データ・ストリームの日付／時刻情報のデータ型として最も一般に使用されるのは、日付と時刻の両方を組み合わせた dateTime です。 ただし、ほとんどの場合、RMDS によって提供され、ロイター・マーケットフィード・アダプタによって Event Stream Processor に送信される更新には、日付と時刻が別の FID として構成されます。

マップ・ファイルは、dateTimeField 要素を提供します。この要素は、この矛盾を修復するために、日付と時刻に対して個別の属性を提供します。 これによって、2 つの FID (1 つは日付、1 つは時刻) をストリーム定義の同じカラムにマップできます。

これら 3 つの属性のいずれかを必ず使用してください。 **dateTime** を使用する場合、単独で使用する必要があります。 **dateName** と **timeName** は、個別に、または一緒に使用できます。

各 FID の値は、Reuters 側の設定ファイルで参照されている FID リストにリストされているものと一致する必要があります (アダプタと共に提供されている FID リストの名前は appendix_a です)。 このファイルは、設定ファイル `rfasub.cfg` で参照されます。

例
```
<streamMap name="stream1">
        <itemName key="true"/>
        <FIDListField />
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <dataField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

この例は、TIMACT と ACTIV_DATE の FID を組み合わせて、Event Stream Processor のソース・ストリーム stream1 の 9 番目のカラムにマップします。

FIDListField
streamMap 定義で、**FIDListField** 要素は、イベントのすべてのロイター FID とそれらの値を Event Stream Processor のソース・ストリームにマップします。

まとめ
```
adapter                    (required, limit one)
  |----publication                    (required, limit one)
```

```
  |----streamMaps                    (required, limit one)
  |       '----streamMap             (required)
  |              |----itemName       (required, limit one)
  |              |----serviceName    (optional)
  |              |----sequenceNumber (optional)
  |              |----itemStale      (optional)
  |              |----dataField      (required)
  |              |----updateNumber   (required)
  |              |----dateTimeField  (optional)
  |              |----FIDListField   (optional)
  |              '----nullField      (optional)
  |----recordTypeMap                 (optional)
  |       '----recordType            (optional)
  |----rfa                           (required, limit one)
  '----itemLists                     (required, limit one)
         '----itemList               (required)
                '----item            (optional)
```

親
streamMap

子
なし

属性

名前	説明	稼働条件
name	アダプタに関連したログ・エントリに表示される文字列	オプション

注意
なし

例

```
<streamMap name="stream1">
        <itemName key="true"/>
        <FIDListField />
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <dataField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

この例で、ソース・ストリームの 2 番目のカラムは、アダプタからの任意の更新の FIDList 文字列を伝達するものとして識別されます。

item
item 要素は、ロイター・マーケットフィード・アダプタがサブスクライブする RIC を識別するために使用されます。

まとめ

```
adapter                    (required, limit one)
  |----publication                    (required, limit one)
  |----streamMaps                     (required, limit one)
  |       '----streamMap              (required)
  |              |----itemName        (required, limit one)
  |              |----serviceName     (optional)
  |              |----sequenceNumber  (optional)
  |              |----itemStale       (optional)
  |              |----dataField       (required)
  |              |----updateNumber    (required)
  |              |----dateTimeField   (optional)
  |              |----FIDListField    (optional)
  |              '----nullField       (optional)
  |----recordTypeMap                  (optional)
  |       '----recordType             (optional)
  |----rfa                            (required, limit one)
  '----itemLists                      (required, limit one)
          '----itemList               (required)
                 '----item            (optional)
```

親
itemList

子
なし

属性

名前	説明	稼働条件
name	アダプタがサブスクライブする RIC。	必須
rfaQueue	指定する場合、デフォルトの rfaQueue の名前に置き換わる rfaQueue の名前。この名前によって、個別のスレッドがこのキューに対して使用されます。	オプション
service	RMDS を介して受信データを提供する Reuters Service の名前。	既に親の itemList 要素または itemLists 要素で指定している場合はオプション、それ以外は必須

名前	説明	稼働条件
stream	この RIC の更新が Event Stream Processor に渡されるソース・ストリーム。	既に親の itemList 要素または itemLists 要素で指定している場合はオプション、それ以外は必須

注意
name 属性の値は、Reuters 側の設定ファイルで参照されている `appendix_a` ファイルにリストされているものと一致する必要があります。アダプタと共に提供されている設定ファイルの名前は、`rfasub.cfg` です。

ストリーム名をここに指定した場合、この RIC の更新は、そのストリームで Event Stream Processor に渡されます。 ここでストリームを指定しない場合、itemList レベルで指定したストリームが使用されます。

指定するストリームは、マップ・ファイルの他の場所で streamMap の name 属性の値を使用して定義された streamMap の 1 つと一致している必要があります。

例
```
<itemLists service="SSL_PUB" stream="stream1">
        <itemList service="IDN_RDF" >
                <item name="EUR=" />
                <item name="EURJPY=" stream="stream6" />
        </itemList>
</itemLists>
```

これらの 2 つの **item** 要素は、アダプタを RIC の EUR と EURJPY にサブスクライブします。 EUR の更新は、itemLists 要素で設定されているストリーム stream1 に送信されます。 EURJPY の更新は、ストリーム stream6 に送信されます。これは、**item** レベルのストリーム属性が、**itemLists** レベルの属性を上書きするためです。

itemList
itemList 要素には、**item** 要素の 1 つ以上のインスタンスがあります。

まとめ
```
adapter                     (required, limit one)
  |----publication                    (required, limit one)
  |----streamMaps                     (required, limit one)
  |      '----streamMap               (required)
  |             |----itemName         (required, limit one)
  |             |----serviceName      (optional)
  |             |----sequenceNumber   (optional)
  |             |----itemStale        (optional)
  |             |----dataField        (required)
  |             |----updateNumber     (required)
```

```
  |             |----dateTimeField   (optional)
  |             |----FIDListField    (optional)
  |             '----nullField       (optional)
  |----recordTypeMap                 (optional)
  |       '----recordType            (optional)
  |----rfa                           (required, limit one)
  '----itemLists                     (required, limit one)
         '----itemList               (required)
                '----item            (optional)
```

親
itemLists

子

名前	稼働条件
item	0 個以上が必要

属性

名前	説明	稼働条件
rfaQueue	指定する場合、デフォルトの rfaQueue の名前に置き換わる rfaQueue の名前。この名前によって、個別のスレッドがこのキューに対して使用されます。	オプション
service	RMDS を介して受信データを提供する Reuters Service の名前。	既に親の itemList 要素またはすべての子 item 要素で指定している場合はオプション、それ以外は必須
stream	このリストの項目に指定した RIC で更新を受信する Event Stream Processor のソース・ストリームの名前。	既に親の itemList 要素またはすべての子 item 要素で指定している場合はオプション、それ以外は必須

注意
この要素のストリーム名を指定してアダプタを設定し、このセクションにある項目ごとの更新をそのストリームにプッシュします (この指定は項目レベルで上書きできます)。

アダプタでは、itemLists の下に複数の itemList 要素を設定できます。このため、アダプタの 1 つのインスタンスが複数の RIC グループからの更新を異なる Event Stream Processor のソース・ストリームに送信するように設定できます。

指定するストリームは、マップ・ファイルの他の場所で streamMap の name 属性の値を使用して定義された streamMap の 1 つと一致している必要があります。

rfaQueue 属性を使用することによって、スケーラビリティを制御します。

例

```
<itemLists service="SSL_PUB" stream="stream1">
        <itemList service="IDN_RDF" >
                <item name="EUR=" />
                <item name="EURJPY=" stream="stream6" />
        </itemList>
</itemLists>
```

この itemList 要素は service 属性を IDN_RDF に設定し、親 itemLists 要素で定義されている SSL_PUB service 属性を上書きします。

itemLists

itemLists 要素には、**itemList** 要素の 1 つ以上のインスタンスがあります。

まとめ

```
adapter                    (required, limit one)
  |----publication                    (required, limit one)
  |----streamMaps                     (required, limit one)
  |       '----streamMap              (required)
  |              |----itemName        (required, limit one)
  |              |----serviceName     (optional)
  |              |----sequenceNumber  (optional)
  |              |----itemStale       (optional)
  |              |----dataField       (required)
  |              |----updateNumber    (required)
  |              |----dateTimeField   (optional)
  |              |----FIDListField    (optional)
  |              '----nullField       (optional)
  |----recordTypeMap                  (optional)
  |       '----recordType             (optional)
  |----rfa                            (required, limit one)
  '----itemLists                      (required, limit one)
         '----itemList                (required)
                '----item             (optional)
```

親

adapter

子

名前	稼働条件
itemList	1 つは必須で、複数もサポート

属性

名前	説明	稼働条件
name	アダプタに関連したログ・エントリに表示される文字列。	オプション
rfaQueue	指定する場合、デフォルトの rfaQueue の名前に置き換わる rfaQueue の名前。この名前によって、個別のスレッドがこのキューに対して使用されます。	オプション
service	RMDS を介して受信データを提供する Reuters サービスの名前。	すべての子 item 要素が指定または継承するように、子 itemLists 要素か item 要素またはその両方が指定されている場合はオプションで、それ以外は必須
stream	このセクションの item リストで指定されている RIC で更新を受信する Event Stream Processor のソース・ストリームの名前。デフォルトは item レベルで上書きできます。	すべての子 item 要素が指定または継承するように、子 itemLists 要素か item 要素またはその両方が指定されている場合はオプションで、それ以外は必須

注意
このセクションの各 **itemList** インスタンスは、アダプタがサブスクライブする 1 つ以上の RIC のリストです。

例

```
<itemLists service="SSL_PUB" stream="stream1">
        <itemList service="IDN_RDF" >
                <item name="EUR=" />
                <item name="EURJPY=" stream="stream6" />
        </itemList>
</itemLists>
```

この itemLists 要素は、service 属性を SSL_PUB に、stream 属性を stream1 に設定します。 これらの属性は、**itemList** レベルか **item** レベル、またはその両方で継承または上書きされます。

itemName
streamMap 定義で、**itemName** 要素が、RMDS 更新から RIC を送る Event Stream Processor のソース・ストリームのローを識別します。

まとめ
```
adapter                    (required, limit one)
  |----publication                    (required, limit one)
  |----streamMaps                     (required, limit one)
  |       '----streamMap              (required)
  |              |----itemName        (required, limit one)
  |              |----serviceName     (optional)
  |              |----sequenceNumber  (optional)
  |              |----itemStale       (optional)
  |              |----dataField       (required)
  |              |----updateNumber    (required)
  |              |----dateTimeField   (optional)
  |              |----FIDListField    (optional)
  |              '----nullField       (optional)
  |----recordTypeMap                  (optional)
  |       '----recordType             (optional)
  |----rfa                            (required, limit one)
  '----itemLists                      (required, limit one)
         '----itemList                (required)
                '----item             (optional)
```

親
streamMap

子
なし

属性

名前	説明	稼働条件
key	true または false (このカラムがソース・ストリームの一意なキーの一部であるかどうかによって決定)	最初の「注意」を参照

注意
key 属性を使用する必要はありません。 下位互換性があります。

itemName 要素は、RIC または証券コードを伝達する RowDef のカラムに対応するために、**streamMap** に挿入する必要があります。 このカラムがソース・ストリームのキーを構成する場合、key 属性を true に設定する必要があります。

この要素は、RMDS から直接受信するデータ・フィードの一部ではないデータ項目を指定する「疑似フィールド」の 1 つです。

例

```
<streamMap name="stream1">
        <itemName key="true"/>
        <FIDListField />
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <dataField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

この例で、ソース・ストリームの最初のカラムは、アダプタからの任意の更新の RIC 値を伝達するものとして識別されます。 ストリームのキーの一部としても識別されます。

itemStale

streamMap 定義で、**itemStale** 要素は、受信 RMDS データが失効したかどうかを示すフラグを伝達する、Event Stream Processor のソース・ストリームのカラムを識別します。

まとめ

```
adapter                    (required, limit one)
  |----publication                   (required, limit one)
  |----streamMaps                    (required, limit one)
  |       '----streamMap             (required)
  |              |----itemName       (required, limit one)
  |              |----serviceName    (optional)
  |              |----sequenceNumber (optional)
  |              |----itemStale      (optional)
  |              |----dataField      (required)
  |              |----updateNumber   (required)
  |              |----dateTimeField  (optional)
  |              |----FIDListField   (optional)
  |              '----nullField      (optional)
  |----recordTypeMap                 (optional)
  |       '----recordType            (optional)
  |----rfa                           (required, limit one)
  '----itemLists                     (required, limit one)
         '----itemList               (required)
                '----item            (optional)
```

親

streamMap

子

なし

属性

名前	説明	稼働条件
name	アダプタに関連したログ・エントリに表示される文字列	必須

注意
ソース・ストリームのカラムのいずれかが「失効」フラグの場合は、この要素を streamMap で使用します。

RMDS 自体は「失効」フラグを通常のマーケット・データに付加しませんが、RMDS を介してサブスクライブしている別のサービスによって付加される場合には、そのまま送信します。 この要素が streamMap で使用されると、アダプタは、RMDS から「失効」フラグを受信した場合に更新値 "1" を送信するか、RMDS からのすべてのデータ受信を停止します。

例

```
<streamMap name="stream1">
        <itemName key="true"/>
        <FIDListField />
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <dataField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

この例では、ソース・ストリームの 4 番目のカラムは、アダプタが「失効」通知を受信するか、RMDS からのデータ受信を停止した場合に、更新されるカラムとして識別されます。

nullField

streamMap の **nullField** 要素は、常に null 値をストリームに配信するプレースホルダとして機能します。 これによって、必要な設定を行うために、ソース・ストリームに余分なフィールドを追加できます。

まとめ

```
adapter                    (required, limit one)
  |----publication                   (required, limit one)
  |----streamMaps                    (required, limit one)
  |      '----streamMap              (required)
  |             |----itemName        (required, limit one)
  |             |----serviceName     (optional)
  |             |----sequenceNumber  (optional)
```

```
  |             |----itemStale       (optional)
  |             |----dataField       (required)
  |             |----updateNumber    (required)
  |             |----dateTimeField   (optional)
  |             |----FIDListField    (optional)
  |             '----nullField       (optional)
  |----recordTypeMap                 (optional)
  |      '----recordType             (optional)
  |----rfa                           (required, limit one)
  '----itemLists                     (required, limit one)
         '----itemList               (required)
                '----item            (optional)
```

親
streamMap

子
なし

属性

名前	説明	稼働条件
name	アダプタに関連したログ・エントリに表示される文字列	オプション

注意
プロジェクトを試行する場合、**nullField** を使用して、データがストリームの 1 つのカラムに配信されるのを一時的に停止できます。 この場合、次の例のように、一時的に置き換える **dataField** の名前を維持できます。

例

```
<streamMap name="stream1">
        <itemName key="true"/>
        <FIDListField />
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <nullField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

この例では、ソース・ストリームの 7 番目のカラムは、アダプタからの各更新で null 値を受信するプレースホルダとして識別されます。 これには、デバッグのために置換される **dataField** の名前があります。

publication
publication 要素は、アダプタのこのインスタンスの基本動作パラメータを指定します。

まとめ

```
adapter                    (required, limit one)
  |----publication                    (required, limit one)
  |----streamMaps                     (required, limit one)
  |      '----streamMap               (required)
  |             |----itemName         (required, limit one)
  |             |----serviceName      (optional)
  |             |----sequenceNumber   (optional)
  |             |----itemStale        (optional)
  |             |----dataField        (required)
  |             |----updateNumber     (required)
  |             |----dateTimeField    (optional)
  |             |----FIDListField     (optional)
  |             '----nullField        (optional)
  |----recordTypeMap                  (optional)
  |      '----recordType              (optional)
  |----rfa                            (required, limit one)
  '----itemLists                      (required, limit one)
         '----itemList                (required)
                '----item             (optional)
```

親
adapter

子
なし

属性

名前	説明	稼働条件
flushInterval	アダプタが待機するマイクロ秒数を指定する。アダプタはこの時間が経過するまでイベントを累積し、経過すると、Event Stream Processor に送信します。 ゼロ以外の値を flushInterval に設定した場合、時間に基づいてイベントを累積します。	オプション
intraSubscribeDelay	アダプタがサブスクリプション要求後に次の要求まで待機するマイクロ秒数を指定する。	オプション

名前	説明	稼働条件
maxRecordsPerBlock	アダプタが一度に Event Stream Processor に送信する累積イベントの最大数を指定する。 これによって、イベントが大量に累積されたときに、トランザクションまたはエンベロープのそれぞれのフラグメントのサイズが小さくなります。 たとえば、maxRecordsPerBlock の値が 50 に設定されて、140 個のイベントが累積された場合、アダプタはエンベロープまたはトランザクションを 3 つのフラグメントとして送信します。	オプション
name	ログ・ファイル・エントリ内でアダプタ・インスタンスを識別する文字列を指定する。	必須
pendingLimit	アダプタが累積するイベント数を指定する。この数に達すると、アダプタは Event Stream Processor にイベントを送信します。 pendingLimit を使用すると、イベントはカウント数に基づいてイベントを累積します。	オプション
retryInterval	アダプタが RMDS に接続を試行する秒数を指定する。待機時間がこの秒数に達すると、アダプタはシャットダウンします。	必須
sendAsTransactions	更新のグループを 1 つのトランザクションとして扱う場合は true に設定する。1 つのエンベロープ内の個別のトランザクションとして扱う場合は false に設定します。	オプション

注意
アダプタのパフォーマンスを最適化できます。これを行うには、**pendingLimit** 属性と **flushInterval** 属性を、アダプタが Event Stream Processor と通信するために使用するパブリッシュ／サブスクライブのインタフェースからの **maxRecordsPerBlock** 属性と **sendAsTransactions** 属性と一緒に使用します。

取引所によっては、マルチパートのメッセージとして初期イメージが送信され、大きいデータ・セットが生成されることがあります。 intraSubscribeDelay 属性を使用すると、これらのサブスクリプションのペースを維持し、アダプタが初期イメージに対応できなくなるのを防止できます。 デフォルト値は 0 です。この値は、短い RIC リストに適しています。 0 以外の値に設定されると、アダプタはサブスクリプション要求後に次の要求まで設定したミリ秒間待機します。 推奨値は 10 です。

例

```
<publication name="RMDS Adapter - low latency" retryInterval="5"
        flushInterval="0" pendingLimit="0" sendAsTransactions="0" />
```

recordType

recordType 要素は、ストリームを事前に定義された FID セットにマップします。

まとめ

```
adapter                     (required, limit one)
  |----publication                    (required, limit one)
  |----streamMaps                     (required, limit one)
  |       '----streamMap              (required)
  |              |----itemName        (required, limit one)
  |              |----serviceName     (optional)
  |              |----sequenceNumber  (optional)
  |              |----itemStale       (optional)
  |              |----dataField       (required)
  |              |----updateNumber    (required)
  |              |----dateTimeField   (optional)
  |              |----FIDListField    (optional)
  |              '----nullField       (optional)
  |----recordTypeMap                  (optional)
  |       '----recordType             (optional)
  |----rfa                            (required, limit one)
  '----itemLists                      (required, limit one)
         '----itemList                (required)
                '----item             (optional)
```

親

recordTypeMap

子

なし

属性

名前	説明	稼働条件
number	Reuters 設定で定義されている recordType の ID	必須
stream	このレコードがマップされるストリームの名前	必須

注意

recordType によって指定されている、事前に定義されたレコードは、ストリームで定義されているすべてのカラムに一致する必要があります。

そうでない場合、これらのカラムは **streamMap** 設定内で明示的にマップされる必要があります。

例

```
<recordTypeMap>
        <recordType number="123" stream="eqInput"/>
</recordTypeMap>
```

この例では、"123" として事前に定義された FID のセットをソース・ストリーム eqInput にマップします。

recordTypeMap

recordTypeMap 要素には、1 つ以上の **recordType** 要素があります。

まとめ

```
adapter                    (required, limit one)
  |----publication                    (required, limit one)
  |----streamMaps                     (required, limit one)
  |       '----streamMap              (required)
  |              |----itemName        (required, limit one)
  |              |----serviceName     (optional)
  |              |----sequenceNumber  (optional)
  |              |----itemStale       (optional)
  |              |----dataField       (required)
  |              |----updateNumber    (required)
  |              |----dateTimeField   (optional)
  |              |----FIDListField    (optional)
  |              '----nullField       (optional)
  |----recordTypeMap                  (optional)
  |       '----recordType             (optional)
  |----rfa                            (required, limit one)
  '----itemLists                      (required, limit one)
         '----itemList                (required)
                '----item             (optional)
```

親

adapter

子

名前	稼働条件
recordType	0 個以上をサポート

属性

なし

注意
ストリームには、recordTypeMap または streamMap のいずれかが設定される必要があります。両方は設定できません。

recordTypeMap によって提供される暗黙的なマッピングを使用するには、事前に定義されたレコードは、ストリームの定義のすべてのカラムに一致する必要があります。

そうでない場合、これらのカラムは streamMap 設定内で明示的にマップされる必要があります。

例
```
<recordTypeMap>
       <recordType number="123" stream="eqInput"/>
</recordTypeMap>
```

この例では、"123" として事前に定義された FID のセットをソース・ストリーム eqInput にマップします。

rfa
rfa 要素は、入力アダプタ・マップ・ファイルを Reuters 側の設定ファイルにリンクします。

まとめ
```
adapter                    (required, limit one)
  |----publication                    (required, limit one)
  |----streamMaps                     (required, limit one)
  |      '----streamMap               (required)
  |             |----itemName         (required, limit one)
  |             |----serviceName      (optional)
  |             |----sequenceNumber   (optional)
  |             |----itemStale        (optional)
  |             |----dataField        (required)
  |             |----updateNumber     (required)
  |             |----dateTimeField    (optional)
  |             |----FIDListField     (optional)
  |             '----nullField        (optional)
  |----recordTypeMap                  (optional)
  |      '----recordType              (optional)
  |----rfa                            (required, limit one)
  '----itemLists                      (required, limit one)
         '----itemList                (required)
                '----item             (optional)
```

親
adapter

子
なし

属性

名前	説明	稼働条件
config	サブスクリプションのための Reuters 側の設定ファイルの絶対パスとファイル名 (アダプタと共に提供されるサンプル・ファイルは $ESP_REUTERS_HOME/config/rfasub.cfg)。	必須
configDatabaseName	RFA に設定する必要あり。	必須
enumFile	各列挙型とその値の範囲をリストする、Reuters 提供のファイルのフル・パス名。	必須
fidFile	有効な FID をすべてリストした、Reuters 提供のファイルのフル・パス名。	必須
sessionName	サブスクリプション用に Reuters 側の設定ファイルで定義されているセッション名を参照。	必須
blank	ブランク用に使用するマーカを指定。	オプション
blankInt32	ブランク Int32 フィールド用に使用するマーカを指定。	オプション
blankInt64	ブランク Int64 フィールド用に使用するマーカを指定。	オプション
blankMoney	ブランクの通貨型フィールド用に使用するマーカを指定。	オプション
blankString	ブランクの文字列フィールド用に使用するマーカを指定。	オプション
blankDate	ブランクの日付フィールド用に使用するマーカを指定。	オプション
blankTimestamp	ブランクのタイムスタンプ・フィールド用に使用するマーカを指定。	オプション

注意
なし

例

```
<rfa config="$ESP_REUTERS_HOME/config/rfasub.cfg"
        sessionName="Session1" />
```

この例では、ロイター・マーケットフィード・アダプタは、ファイル `rfasub.cfg` の Reuters 側設定を指し示します。 この設定ファイルのリスト行は、次のとおりです。

```
¥Sessions¥Session1¥connectionList =
"Connection_SSLED"
```

この行は、設定ファイルの他の行によって参照されるセッション名を定義します。 マップ・ファイルは **sessionName** 属性のセッション名を参照するときに、アダプタを、その名前で識別される Reuters 側の設定パラメータにリンクします。

sequenceNumber

streamMap 定義では、**sequenceNumber** 要素が、RMDS からデータの一部として提供される一意の数ではなく、アダプタが生成する一意の数によって設定されるソース・ストリームのカラムをマップします。

まとめ

```
adapter                    (required, limit one)
  |----publication                   (required, limit one)
  |----streamMaps                    (required, limit one)
  |       '----streamMap             (required)
  |              |----itemName       (required, limit one)
  |              |----serviceName    (optional)
  |              |----sequenceNumber (optional)
  |              |----itemStale      (optional)
  |              |----dataField      (required)
  |              |----updateNumber   (required)
  |              |----dateTimeField  (optional)
  |              |----FIDListField   (optional)
  |              '----nullField      (optional)
  |----recordTypeMap                 (optional)
  |       '----recordType            (optional)
  |----rfa                           (required, limit one)
  '----itemLists                     (required, limit one)
         '----itemList               (required)
                '----item            (optional)
```

親
streamMap

子
なし

属性

名前	説明	稼働条件
key	true または false (このカラムがソース・ストリームの一意なキーの一部であるかどうかによって決定)	「注意」を参照
name	ログ・エントリに表示される文字列	オプション

注意
アダプタは、サブスクライブ先の RIC の個別のカウンタを維持します。 RIC の更新を受信するたびに、アダプタはその RIC のカウンタを増やします。 この数値は、sequenceNumber 要素でマップされるソース・ストリームのカラムに送信されます。

多くのソース・ストリーム定義には、以下のようなカラム指定があります。

```
<Column datatype="int32" name="Id"/>
```

この行はソース・ストリームの一意な ID を指定します。 **sequenceNumber** 疑似フィールドは、入力アダプタ・マップ・ファイルにあるこのカラムに適しています。

値を true に設定するには、key 属性を使用する必要があります。 このカラムがストリームのキーの一部ではない場合、この属性は省略できます。

例

```
<streamMap name="stream1">
        <itemName key="true"/>
        <FIDListField />
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <dataField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

この例では、ソース・ストリームの 3 番目のカラムが、アダプタによって提供されるシーケンス番号にマップされています。このカラムはソース・ストリームの一意なキーの一部としても識別されます。

serviceName

streamMap 定義で、**serviceName** 要素は、ソース・ストリームのカラムを、アダプタが各更新の「エンベロープ」の一部として提供するサービス識別子にマップします。

まとめ

```
adapter                    (required, limit one)
  |----publication                    (required, limit one)
  |----streamMaps                     (required, limit one)
  |       '----streamMap              (required)
  |              |----itemName        (required, limit one)
  |              |----serviceName     (optional)
  |              |----sequenceNumber  (optional)
  |              |----itemStale       (optional)
  |              |----dataField       (required)
  |              |----updateNumber    (required)
  |              |----dateTimeField   (optional)
  |              |----FIDListField    (optional)
  |              '----nullField       (optional)
  |----recordTypeMap                  (optional)
  |       '----recordType             (optional)
  |----rfa                            (required, limit one)
  '----itemLists                      (required, limit one)
          '----itemList               (required)
                 '----item            (optional)
```

親

streamMap

子

なし

属性

名前	説明	稼働条件
key	true または false (このカラムがソース・ストリームの一意なキーの一部であるかどうかによって決定)	「注意」を参照

注意

値を true に設定するには、key 属性を使用する必要があります。 このカラムがストリームのキーの一部ではない場合、この属性は省略できます。

例

```
<streamMap name="stream1">
        <itemName key="true"/>
        <FIDListField />
```

```
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <dataField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

この例では、コメントアウトされているため、アダプタによって提供されるサービス名にソース・ストリームのいずれのカラムもマップされません。

streamMap

streamMap 要素には、Event Stream Processor のソース・ストリームのカラムとアダプタによってサブスクライブされる RMDS FID との間のマッピングがあります。

まとめ

```
adapter                    (required, limit one)
  |----publication                    (required, limit one)
  |----streamMaps                     (required, limit one)
  |       '----streamMap              (required)
  |              |----itemName        (required, limit one)
  |              |----serviceName     (optional)
  |              |----sequenceNumber  (optional)
  |              |----itemStale       (optional)
  |              |----dataField       (required)
  |              |----updateNumber    (required)
  |              |----dateTimeField   (optional)
  |              |----FIDListField    (optional)
  |              '----nullField       (optional)
  |----recordTypeMap                  (optional)
  |       '----recordType             (optional)
  |----rfa                            (required, limit one)
  '----itemLists                      (required, limit one)
          '----itemList               (required)
                 '----item            (optional)
```

親

streamMaps

子

以下の子要素は、**streamMap** に対して定義されます。 これらの子要素は、任意の順序で指定できますが、特定の **streamMap** の場合、子要素の順序は、ソース・ストリームのカラムの順序を反映している必要があります (プロジェクトで定義されている順序)。 これによって、ソース・ストリームの適切なローに RMDS 更新を配信するようにアダプタが設定されます。

名前	稼働条件
dataField	1 つは必須で、複数もサポート
dateTimeField	0 個以上をサポート
itemName	1 つは必須で、複数もサポート
itemStale	0 または 1 をサポート
sequenceNumber	0 または 1 をサポート
serviceName	0 または 1 をサポート

属性

名前	説明	稼働条件
name	RMDS 更新のマップ先のソース・ストリームを参照。 Event Stream Processor プロジェクトで定義されているソース・ストリームの名前に一致する必要があります。	必須
opcode	アダプタが更新をソース・ストリームに送信するときに実行するオペレーションを定義。 有効な値は、insert と upsert です。insert オペレーションは、新しい更新をソース・ストリームの末尾に追加します。 upsert オペレーションは、既存のソース・ストリームのエントリをキーが一致する場合に置き換えます。一致しない場合は、更新を追加します。	オプション (デフォルト値は upsert)

注意
なし

例

```
<streamMaps>
        <streamMap name="stream1">
                <itemName key="true"/>
                <FIDListField />
                <!-- serviceName / -->
                <sequenceNumber />
                <itemStale/>
                <dataField name="BID"/>
                <dataField name="ASK"/>
                <dataField name="TRDPRC_1"/>
                <dataField name="ACVOL_1"/>
                <dateTimeField timeName="TIMACT"
dateName="ACTIV_DATE"/>
        </streamMap>
</streamMaps>
```

この例は、アダプタの一連の更新を Event Stream Processor のソース・ストリーム stream1 にマップします。このソース・ストリームに送信されるすべての更新は、upsert モードを使用して追加されます。 その更新がこのソース・ストリームに送信される RIC は、stream1 を参照するマップ・ファイルの別の場所の itemList で指定されます。

streamMaps
streamMaps 要素には、**streamMap** 要素の 1 つ以上のインスタンスがあります。

まとめ

```
adapter                     (required, limit one)
  |----publication                    (required, limit one)
  |----streamMaps                     (required, limit one)
  |       '----streamMap              (required)
  |              |----itemName        (required, limit one)
  |              |----serviceName     (optional)
  |              |----sequenceNumber  (optional)
  |              |----itemStale       (optional)
  |              |----dataField       (required)
  |              |----updateNumber    (required)
  |              |----dateTimeField   (optional)
  |              |----FIDListField    (optional)
  |              '----nullField       (optional)
  |----recordTypeMap                  (optional)
  |       '----recordType             (optional)
  |----rfa                            (required, limit one)
  '----itemLists                      (required, limit one)
         '----itemList                (required)
                '----item             (optional)
```

親
adapter

子

名前	稼働条件
streamMap	1 つは必須で、複数もサポート

属性
なし

注意
このセクションの各 **streamMap** インスタンスは、Reuters アダプタからの受信 FID を Event Stream Processor のソース・ストリームのカラムにマップします。

例

```
<streamMaps>
        <streamMap name="stream1">
                <itemName key="true"/>
                <FIDListField />
                <!-- serviceName / -->
                <sequenceNumber />
                <itemStale/>
                <dataField name="BID"/>
                <dataField name="ASK"/>
                <dataField name="TRDPRC_1"/>
                <dataField name="ACVOL_1"/>
                <dateTimeField timeName="TIMACT"
dateName="ACTIV_DATE"/>
        </streamMap>
</streamMaps>
```

updateNumber

streamMap 定義では、**updateNumber** 要素が、RMDS からデータの一部として提供される一意の数ではなく、アダプタが生成する一意の数によって設定される Event Stream Processor のソース・ストリームのカラムをマップします。

まとめ

```
adapter                    (required, limit one)
  |----publication                    (required, limit one)
  |----streamMaps                     (required, limit one)
  |       '----streamMap              (required)
  |              |----itemName        (required, limit one)
  |              |----serviceName     (optional)
  |              |----sequenceNumber  (optional)
  |              |----itemStale       (optional)
  |              |----dataField       (required)
  |              |----updateNumber    (required)
  |              |----dateTimeField   (optional)
  |              |----FIDListField    (optional)
  |              '----nullField       (optional)
  |----recordTypeMap                  (optional)
  |       '----recordType             (optional)
  |----rfa                            (required, limit one)
  '----itemLists                      (required, limit one)
          '----itemList               (required)
                 '----item            (optional)
```

親
streamMap

子
なし

属性

名前	説明	稼働条件
key	true または false (このカラムがソース・ストリームの一意なキーの一部であるかどうかによって決定)	「注意」を参照
name	ログ・エントリに表示される文字列	オプション

注意

アダプタは、サブスクライブ先の RIC の個別のカウンタを維持します。 RIC の更新を受信するたびに、アダプタはその RIC のカウンタを増やします。 この数値は、**updateNumber** 要素でマップされるソース・ストリームのカラムに送信されます。

多くのソース・ストリーム定義には、以下のようなカラム指定があります。

```
<Column datatype="integer" name="Id"/>
```

この行はソース・ストリームの一意な ID を指定します。 **updateNumber** 疑似フィールドは、入力アダプタ・マップ・ファイルにあるこのカラムに適しています。

値を true に設定するには、key 属性を使用する必要があります。 このカラムがストリームのキーの一部ではない場合、この属性は省略できます。

例

```
<streamMap name="stream1">
        <itemName key="true"/>
        <FIDListField />
        <!-- serviceName / -->
        <updateNumber />
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <dataField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

この例では、ソース・ストリームの 3 番目のカラムが、アダプタによって提供される更新番号にマップされています。このカラムはソース・ストリームの一意なキーの一部としても識別されます。

出力アダプタ・マップ・ファイルの XML 構文

ロイター・マーケットフィード・アウトプット・アダプタ用のマップ・ファイルの構文を以下に示します。

次のリストは、出力アダプタ・マップ・ファイルの構造を示します。 このサマリの各行は、マップ・ファイル構造の 1 つの要素をリストします。 詳細については、各要素のトピックを参照してください。

```
adapter                                 (required, limit one)
  |----rfa                              (required, limit one)
  '----subscriptions                    (required, limit one)
         '----subscription              (required)
                '----stream             (required)
                       |----name        (required, limit one)
                       |----service     (optional)
                       |      '----enum (required)
                       |----stale       (optional)
                       |----field       (required)
                       '----constant    (optional)
```

adapter

adapter 要素は、出力マップ・ファイルのルート要素です。

まとめ


```
adapter                                 (required, limit one)
  |----rfa                              (required, limit one)
  '----subscriptions                    (required, limit one)
         '----subscription              (required)
                '----stream             (required)
                       |----name        (required, limit one)
                       |----service     (optional)
                       |      '----enum (required)
                       |----stale       (optional)
                       |----field       (required)
                       '----constant    (optional)
```


すべての設定要素は、開始と終了の **adapter** タグの間にネストされる必要があります。

親

なし

子

以下の子要素が **adapter** 用に定義されます。 これらのすべての要素は、指定された順序で存在する必要があります。

名前	稼働条件
rfa	1 つだけ必要

名前	稼働条件
サブスクリプション	1 つだけ必要

属性
なし

注意
なし

例
子要素の例を参照してください。

constant
constant 要素は、アダプタによって RMDS にパブリッシュされる定数値で構成されるデータ項目を定義します。

まとめ

```
adapter                                  (required, limit one)
  |----rfa                               (required, limit one)
  '----subscriptions                     (required, limit one)
         '----subscription               (required)
                '----stream              (required)
                       |----name         (required, limit one)
                       |----service      (optional)
                       |      '----enum  (required)
                       |----stale        (optional)
                       |----field        (required)
                       '----constant     (optional)
```

親
stream

子
なし

属性

名前	説明	稼働条件
name	アダプタによってパブリッシュされるイメージ内のこのデータ項目に関連付けられている名前	必須
value	この constant の値 (このデータ項目が RMDS にパブリッシュされる場合は常に同じ)	必須

注意
アダプタは起動時に、完全なイメージを RMDS にパブリッシュします。このイメージには、マップ・ファイルで定義されているすべてのデータ項目があります。その後に変更が発生した場合は、Event Stream Processor が失効して復旧した場合を除いて、アダプタはデータ項目の更新された値のみをパブリッシュします。 これは、**constant** の値がパブリッシュされるのは、完全なイメージがパブリッシュされた場合のみであることを意味します。

例
```
<stream name="stream1" >
        <name column="0"/>
        <stale column="3" name="ACVOL_1"/>
        <field column="1" name="DSPLY_NAME" />
        <field column="4" name="BID" precision="47" />
        <field column="5" name="ASK" precision="0" />
        <field column="6" name="TRDPRC_1"/>
        <field column="7" name="ACVOL_1"/>
        <constant name="PROD_PERM" value="1"/>
</stream>
```

この例は、PROD_PERM と呼ばれる constant を定数値 "1" で定義します。この constant は、パブリッシュ名 subscription1 の下のストリーム stream1 からのデータ値と共にパブリッシュされます。

enum
enum 要素は、Event Stream Processor ストリームのサービス・カラムの値を、アダプタによって RMDS にパブリッシュされる更新の name 要素の前に付加される一意な文字列にマップします。

まとめ
```
adapter                                 (required, limit one)
  |----rfa                              (required, limit one)
  '----subscriptions                    (required, limit one)
         '----subscription              (required)
                '----stream             (required)
                       |----name        (required, limit one)
                       |----service     (optional)
                       |      '----enum (required)
                       |----stale       (optional)
                       |----field       (required)
                       '----constant    (optional)
```

パブリッシュ元の Event Stream Processor ストリームが、異なるソースからの同じ証券コードのデータ項目 (たとえば、NASDAQ からと S&P からの IBM の「売値」価格) を処理する場合、出力アダプタ・マップ・ファイルの service 属性と enum 属

性を使用して、異なるソースからの同じ証券コードの同じ値の更新を区別するようにアダプタを設定できます。

親
service

子
なし

属性

名前	説明	稼働条件
value	**service** 要素で指定されるデータ・ストリーム・カラムの可能性のある値	必須
prefix	**name** 要素が、**prefix** に一致する **service** 値と共に Event Stream Processor から受信された更新をパブリッシュする場合に、この要素の値の前に付加される文字列	必須

注意
出力アダプタ・マップ・ファイルの **service** 要素には、ソース・カラムの可能な値ごとに 1 つの **enum** 要素が構成されている必要があります。

例
```
<service column="2" delim="_">
        <enum value="RDF" prefix="R"/>
        <enum value="ISFS" prefix="I"/>
</service>
```

service 定義内で、各 **enum** 要素は、特定のサービスを指定します。 この値に基づいて、パブリッシュされた RIC の名前は、データのプロバイダを示すために変更されます。 RIC.X が、name カラムで見つかった RIC であると想定すると、 カラム 2 の値が RDF の場合、RIC は "R_RIC.X" になります。 カラム 2 の値が ISFS の場合、RIC は "I_RIC.X" になります。 いずれも真ではない場合、値は何もパブリッシュされません。

field
出力アダプタ・マップ・ファイルの **stream** 定義では、**field** はパブリッシュするストリーム内のカラムを指定します。

まとめ
```
adapter                                    (required, limit one)
  |----rfa                                 (required, limit one)
```

```
  '----subscriptions                      (required, limit one)
         '----subscription                (required)
                '----stream               (required)
                       |----name          (required, limit one)
                       |----service       (optional)
                       |      '----enum   (required)
                       |----stale         (optional)
                       |----field         (required)
                       '----constant      (optional)
```

親
stream

子
なし

属性

名前	説明	稼働条件
column	パブリッシュされるストリーム内のソース・カラムの位置を表す番号 (ストリーム内の最初のカラムの番号は、"0")	必須
name	RMDS にパブリッシュされたときにこのデータ値を識別する FID	必須
precision	パブリッシュされた値内の小数点以下の総桁数を指定する整数 (たとえば、1.23 の precision は 2)	オプション

注意
name 属性の値は、このストリーム定義で と **enum** 要素を定義した場合に、データ項目のソースを示すように変更されることがあります。

precision 属性を指定できるのは、データ型 double のカラムのみです。

例

```
<stream name="stream1" >
        <name column="0"/>
        <stale column="3" name="ACVOL_1"/>
        <field column="1" name="DSPLY_NAME" />
        <field column="4" name="BID" precision="47" />
        <field column="5" name="ASK" precision="0" />
        <field column="6" name="TRDPRC_1"/>
        <field column="7" name="ACVOL_1"/>
        <constant name="PROD_PERM" value="1"/>
</stream>
```

この例では、アダプタは、stream1 と呼ばれる Event Stream Processor ストリームの 4 番目、5 番目、6 番目、7 番目のカラムの更新情報を、それぞれ BID、ASK、

TRDPRC_1、ACVOL_1 の名前のデータ項目としてパブリッシュするように設定されます。

name
出力アダプタ・マップ・ファイルの **stream** 定義では、**name** は、各更新情報を識別するために使用される値を提供する、ソース・ストリーム内のカラムを指定します。

まとめ

```
adapter                                  (required, limit one)
  |----rfa                               (required, limit one)
  '----subscriptions                     (required, limit one)
         '----subscription               (required)
                '----stream              (required)
                       |----name         (required, limit one)
                       |----service      (optional)
                       |      '----enum  (required)
                       |----stale        (optional)
                       |----field        (required)
                       '----constant     (optional)
```

親
stream

子
なし

属性

名前	説明	稼働条件
column	ストリームの一意な識別子を伝達する、ストリーム内のカラムの位置を表す番号 (ストリーム内の最初のカラムの番号は、"0")	カラムまたは名前
name	ストリームの一意な識別子を伝達する、ストリーム内のカラムの名前	カラムまたは名前

注意
出力アダプタは、RMDS を単純なメッセージ・バスとして使用します。パブリッシュされる更新情報は、Reuters プロトコルに準拠する必要はありません。 これは、この要素によって指定されるカラムが Reuters RIC である必要はありませんが、Reuters RIC 構文に従う必要があることを意味します。

ソース・ストリームの一意なキーが 2 つ以上のカラムで構成されている場合は、name 要素を service 要素の 1 つ以上のインスタンスと組み合わせて、完全に一意な名前を使用して更新をパブリッシュするようにアダプタを設定できます。

例

```
<stream name="stream1" >
        <name column="0"/>
        <stale column="3" name="ACVOL_1"/>
        <field column="1" name="DSPLY_NAME" />
        <field column="4" name="BID" precision="47" />
        <field column="5" name="ASK" precision="0" />
        <field column="6" name="TRDPRC_1"/>
        <field column="7" name="ACVOL_1"/>
        <constant name="PROD_PERM" value="1"/>
</stream>
```

この例では、stream1 の最初のカラムが一意な識別子または「キー」カラムとして識別されます。

rfa

rfa 要素は、アダプタの Reuters 側を設定するための情報を提供します。これには、Reuters 側の設定ファイルへの明示的な参照があります。

まとめ

```
adapter                                  (required, limit one)
  |----rfa                               (required, limit one)
  '----subscriptions                     (required, limit one)
         '----subscription               (required)
                '----stream              (required)
                       |----name         (required, limit one)
                       |----service      (optional)
                       |       '----enum (required)
                       |----stale        (optional)
                       |----field        (required)
                       '----constant     (optional)
```

親

adapter

子

なし

属性

名前	説明	稼働条件
serviceName	ロイター・マーケットフィード・アダプタによって送信される各更新のヘッダにあるサービス名を定義	オプション
config	パブリッシュのための Reuters 側の設定ファイルの絶対パスとファイル名 (アダプタと共に提供されるサンプル・ファイルは `$ESP_REUTERS_HOME/config/rfapub.cfg`)	必須
sessionName	パブリッシュ用に Reuters 側の設定ファイルで名前付けされ、定義されているセッションを参照	必須
configDatabaseName	Reuters データベース名への参照	オプション

注意
なし

例

```
<rfa serviceName="IDN_RDF"
        config="$ESP_REUTERS_HOME/config/rfapub.cfg"
        sessionName="Session1" configDatabaseName="RFA" />
```

この例では、ロイター・マーケットフィード・アダプタは、ファイル `rfapub.cfg` の Reuters 側設定を指し示します。この設定ファイルのコメント化されていない最初の 4 行は次のとおりです。

```
¥Connections¥Connection_SSLED_MP¥ipcServerName = "8105"
¥Connections¥Connection_SSLED_MP¥connectionType = "SSLED_MP"
¥Connections¥Connection_SSLED_MP¥entitlementData = false
¥Sessions¥Session1¥connectionList = "Connection_SSLED_MP"
```

これらの行の最後は、マップ・ファイルで **sessionName** として定義されているセッション名を暗黙的に定義しています。 このセッション名に関する `rfapub.cfg` キーから他の 3 つの行。 これは、**sessionName** の値が、マップ・ファイルのこの **publication** セクションを `.cfg` ファイルの設定セットにどのように結びつけているかを示します。

アダプタがこの設定を使用してパブリッシュする場合、各更新情報は、**serviceName** "IDN_RDF" を使用して識別されます。

service
出力アダプタ・マップ・ファイルの **stream** 定義で、**service** はソース・ストリームの一意なキーの別のコンポーネントであるカラムを識別します。

まとめ
```
adapter                                 (required, limit one)
  |----rfa                              (required, limit one)
  '----subscriptions                    (required, limit one)
         '----subscription              (required)
                '----stream             (required)
                       |----name        (required, limit one)
                       |----service     (optional)
                       |      '----enum (required)
                       |----stale       (optional)
                       |----field       (required)
                       '----constant    (optional)
```

親
stream

子
なし

属性

名前	説明	稼働条件
column	セカンダリ・キー値を持つカラムの位置を表す番号 (ストリーム内の最初のカラムの番号は、"0")	必須
delim	名前とプレフィックスとの間の区切りとして使用される文字を指定	オプション

注意
出力アダプタのマップ・ファイルの **service** 要素には、ソース・カラムの可能な値ごとに 1 つの **enum** 要素が構成されている必要があります。

例
```
<service column="2" delim="_">
        <enum value="RDF" prefix="R"/>
        <enum value="ISFS" prefix="I"/>
</service>
```

このセクションは、アダプタが Event StreamProcessor ストリームからの各更新の 2 番目のカラムの値をテストする方法を設定します。2 番目のカラムは、**stream** 要素の **name** 属性です。

値が "RDF" の場合、アダプタはプレフィックス "R" とその後に指定された delim 値を、発行された更新の名前に追加します。更新の名前は、**publication** 要素の **name** 属性の値です。

値が "ISFS" の場合、アダプタはプレフィックス "I" をパブリッシュされた更新の **name** に追加します。

stale
出力アダプタのマップ・ファイルの **stream** 定義で、**stale** 要素は、ストリームが失効すると値が "0" から "1" に変化する、ソース・ストリームのカラムを識別します。

まとめ

```
adapter                                  (required, limit one)
  |----rfa                               (required, limit one)
  '----subscriptions                     (required, limit one)
         '----subscription               (required)
                '----stream              (required)
                       |----name         (required, limit one)
                       |----service      (optional)
                       |      '----enum  (required)
                       |----stale        (optional)
                       |----field        (required)
                       '----constant     (optional)
```

たとえば、ストリーム内のデータ・ソースの 1 つがそれ以上更新されない場合、ストリームは「失効」に変更したと見なされます。

親
stream

子
なし

属性

名前	説明	稼働条件
column	セカンダリ・キー値を持つカラムの位置を表す番号 (ストリーム内の最初のカラムの番号は、"0")	必須
name	FID にマップ (パブリッシュ) できるように、失効カラムを識別する文字列	オプション

注意
なし

例

```
<stream name="stream1" >
        <name column="0"/>
        <stale column="3" name="ACVOL_1"/>
        <field column="1" name="DSPLY_NAME" />
        <field column="4" name="BID" precision="47" />
        <field column="5" name="ASK" precision="0" />
        <field column="6" name="TRDPRC_1"/>
        <field column="7" name="ACVOL_1"/>
        <constant name="PROD_PERM" value="1"/>
</stream>
```

この例では、stream1 の 3 番目のカラムが「失効」カラムとして識別されます。「失効」カラムが指定された場合、そのカラムの値はパブリッシュされ、RIC は失効としてマーク付けされます。

stream

出力アダプタのマップ・ファイルの [subscription] セクションで、アダプタがデータを取得するストリームを識別します。このデータは RMDS にパブリッシュされます。

まとめ

```
adapter                                 (required, limit one)
  |----rfa                              (required, limit one)
  '----subscriptions                    (required, limit one)
         '----subscription              (required)
                '----stream             (required)
                       |----name        (required, limit one)
                       |----service     (optional)
                       |      '----enum (required)
                       |----stale       (optional)
                       |----field       (required)
                       '----constant    (optional)
```

親

subscription

子

名前	稼働条件
name	1 つ
service	オプション
stale	オプション
field	1 つ以上

名前	稼働条件
constant	オプション

属性

名前	説明	稼働条件
exitOnStreamExit	ブール属性。 true の場合、ストリームまたは Event Stream Processor が存在するか、接続が失われると、rmds は終了します。	オプション
finalizer	この文字列は、イベントが Event Stream Processor に 1 つもパブリッシュされずに heartbeat で指定されたミリ秒が経過した場合に実行するアクションを指定する。	オプション
heartbeat	この整数は、ミリ秒単位の待機時間を指定する。イベントが Event Stream Processor に 1 つもパブリッシュされない状態がこの時間を過ぎて継続すると、finalizer アクションが実行されます。	オプション
name	アダプタがデータを受信するストリーム名。このデータは、RMDS でパブリッシュされます。	必須
templateNumber	RMDS 設定での Reuters テンプレート設定。	オプション

注意

name 属性の値は、Event Stream Processor プロジェクトで定義する必要があります。

Event Stream Processor プロジェクトのストリームはいずれも、マップ・ファイルの 1 つの **stream** セクションにのみマップされます。

templateNumber は、それが定義されているストリームの一意な識別子である必要があります。

例

```
<stream name="stream1">
        <name column="0"/>
        <field column="4" name="TRDPRC_1"/>
        <field column="9" name="BID" precision="5"/>
</stream>
```

この例では、名前 stream1 のストリームからデータをパブリッシュするように Event Stream Processor が設定されています。

サブスクリプション

subscription 要素には、**stream** 要素の 1 つ以上のインスタンスが含まれています。これによって、1 つ以上のストリームからデータを受信するようにアダプタを設定できます。

まとめ

```
adapter                                  (required, limit one)
  |----rfa                               (required, limit one)
  '----subscriptions                     (required, limit one)
         '----subscription               (required)
                '----stream              (required)
                       |----name         (required, limit one)
                       |----service      (optional)
                       |      '----enum  (required)
                       |----stale        (optional)
                       |----field        (required)
                       '----constant     (optional)
```

出力アダプタのマップ・ファイルには、複数の **subscription** セクションを構成できます。 実行時に、各 **subscription** セクションに対するパブリッシュ・メカニズムが個別のスレッドにインスタンス化されます。これによって、スケーラビリティが確保されます。

親

subscriptions

子

名前	稼働条件
stream	1 つ以上

属性

名前	説明	稼働条件
name	RMDS でパブリッシュされた更新とログ・ファイル・エントリに表示されるこのサブスクリプションの名前	必須

注意

なし

例

```
<subscriptions>
    <subscription name="subscription1" >
        <stream name="stream1" >
            <name column="0"/>
```

```
            <field column="4" name="BID"/>
            <field column="5" name="ASK"/>
            <field column="6" name="TRDPRC_1"/>
            <field column="7" name="ACVOL_1"/>
            <constant name="PROD_PERM" value="1"/>
        </stream>
    </subscription>
</subscriptions>
```

この例は、Event Stream Processor ストリーム stream1 からのいくつかのカラムを名前 subscription1 を使用してパブリッシュするようにアダプタを設定します。

subscriptions
subscriptions 要素には、1 つ以上の **subscription** 要素があります。

まとめ

```
adapter                                  (required, limit one)
  |----rfa                               (required, limit one)
  '----subscriptions                     (required, limit one)
         '----subscription               (required)
                '----stream              (required)
                       |----name         (required, limit one)
                       |----service      (optional)
                       |      '----enum  (required)
                       |----stale        (optional)
                       |----field        (required)
                       '----constant     (optional)
```

親
adapter

子

名前	稼働条件
subscription	1 つ以上

属性
なし

注意
このセクションの各 **subscription** インスタンスは、アダプタが RMDS にパブリッシュするデータの 1 つのセットを定義します。

例
個別の **subscription** インスタンスの例を参照してください。

ロギング機能

ロイター・マーケットフィード・アダプタは、2 つの異なるロギング・メカニズムをサポートします。

独自のロギング・メカニズムに加えて、ロイター・マーケットフィード・アダプタは、Reuters 側のロギングを利用できます。 これらの両方とも、アダプタのパフォーマンスをチェックし、問題を診断するために使用できます。

これらのログを stderr、syslog、または両方に書き込むように設定できます。

アダプタのロギング

ロイター・マーケットフィード・アダプタは、Event Stream Processor と同じロギングのオプションをサポートします。

-d オプションは、デバッグ・レベルを設定します (0= 緊急メッセージのみ、7= すべてのメッセージ)。

-l オプションは、アダプタにログ・メッセージを stderr、syslog、または両方のいずれかに書き込むように指定します。いずれにも書き込まないようにも指定できます。 **-l** オプションを使用してアダプタにログ・メッセージを stderr に書き込むように指示する場合、stderr をファイルにリダイレクトすることをおすすめします。

入力アダプタ・マップ・ファイルの **publication** 要素の name 属性は、アダプタがどのように設定されているかを容易に識別できるようにするためにログ記録される説明的なテキスト文字列を指定します。たとえば、`subexample.xml` の行 3 〜 6 は、以下のように、ロイター・マーケットフィード・アダプタのサブスクライブ元インスタンスの **publication** 要素を指定します。

```
<publication
    name="RMDS Adapter exp"
    retryInterval="5"
/>
```

アダプタは Event Stream Processor に接続して対話するので、この設定を行うと、アダプタはログ・メッセージを以下のように書き込みます。

```
(0.123) @1 INFO: Configuring publication with name RMDS Adapter exp
```

最初の 2 つのフィールドは、それぞれタイムスタンプ (起動してからの秒数) とスレッド番号です。 タイムスタンプの基本時間と他の情報が、以下の例に示されているように、起動時にログ・ファイルに書き込まれます。 タイムスタンプを日付と時刻に変換するには、基本時間に秒数を単純に加算します。 これらのフィールドは、「ログ・メッセージ」トピックのサンプル・メッセージにはありません。

```
(63359098041.768) @1 NOTICE:Base time is 10/08/08-17:27:21
(0.001) @1 NOTICE:insta-a sub -c cimtest:-- -d 7
-f /home/sybase/support/1.0.3/ReutersAdapter/quotes.map.xml
-l 1 -p tigris:12192 -P 1
```

```
(0.001) @1 NOTICE:pid=28649
(0.001) @1 DEBUG:Using ESP_RMDS_SUBSCRIBE_DEBUG_LEVEL=7ll/
i86pc_64_spro/bin/rmds version:
1.0.3a-alpha_r18674M
```

ページ・データと部分的なページ更新

一部の Reuters データは、Marketfeed Partial フォーマットを使用するページとして到着します。 各ページは、複数行で構成されており、最初はスナップショットとして送信されます。 ページ・データは、いずれの特別な設定も必要とせずにサポートされます。 アダプタ・ログ・ファイルからの以下の引用は、最初のページ・イメージ (強調表示されています) の配信を示しています。

```
(27.729) @6 INFO:Publishing VOD.mGBPd 21 of 21 on stream1 as UPSERT
_ITEM_NAME_ STRING: VOD.mGBPd
_SERVICE_NAME_ STRING: IDN_RDF
_SEQUENCE_NUMBER_ INTEGER: 1
_ITEM_STALE_ INTEGER: 0
ROW80_1 STRING: VOD.mGBPd SI Quote Publication
ROW80_2 STRING:
ROW80_3 STRING: DATE:03/07/2008 Time:11:09
ROW80_4 STRING:
ROW80_5 STRING: Time Venue SI Bid Size Bid Price Ask Price Ask Size
Status
ROW80_6 STRING: ==== ===== == ======== ========= ========= ========
======
ROW80_7 STRING: 110937 GSILGB2XXXX GSIL 1 150.9000 150.9500 1 OPEN
ROW80_8 STRING: 070021 SBILGB2LXXX CITI OPEN
ROW80_9 STRING: 110909 CSFBGB2LXXX CSFB 329 150.7000 151.1500 329
OPEN
ROW80_10 STRING: 110942 DEUTGB22ZEQ DBBL 528 150.6500 151.2000 527
OPEN
ROW80_11 STRING: 110946 ABNAGB22XXX ABNV 483306 150.9000 150.9500
483306 OPEN
ROW80_12 STRING: 110936 UBSWGB2LEQU UBSI 1 149.7682 152.1325 1 OPEN
ROW80_13 STRING: 110828 SBUKGB21XXX CITI 20600 150.9000 151.0000
20600 OPEN
ROW80_14 STRING: 110937 SLIIGB2LXXX LEHM 3750 150.9000 150.9500 15
OPEN
ROW80_15 STRING:
ROW80_16 STRING:
ROW80_17 STRING:
(27.730) @6 DEBUG:Immediate flush for low latency; opcode=p
```

ページの各行には、独自の FID があり、ページに対する行指向のデルタを促進します。 アダプタは、Reuters からの部分的なページ更新を解析し、アダプタ・ログ・ファイルからの以下の引用で強調表示されているような文字列を生成します。

```
(49.934) @6 DEBUG:Processing update for VOD.mGBPd from service
IDN_RDF
(49.934) @6 INFO:Publishing VOD.mGBPd 4 of 21 on stream1 as UPSERT
_ITEM_NAME_ STRING: VOD.mGBPd
_SEQUENCE_NUMBER_ INTEGER: 2
ROW80_3 STRING: off:78 size:2 value:10
```

```
ROW80_11 STRING: off:2 size:3 value:101
(49.934) @6 DEBUG:Immediate flush for low latency; opcode=p
(50.315) @6 DEBUG:Processing update for VOD.mGBPd from service
IDN_RDF
(50.315) @6 INFO:Publishing VOD.mGBPd 3 of 21 on stream1 as UPSERT
_ITEM_NAME_ STRING: VOD.mGBPd
_SEQUENCE_NUMBER_ INTEGER: 3
ROW80_11 STRING: off:5 size:1 value:7
(50.315) @6 DEBUG:Immediate flush for low latency; opcode=p
```

この例の最初の更新は、2 文字の文字列 "10" を、ROW80_3 FID からのデータで構成されるページの行の 78 文字目のオフセットに書き込みます。 例の 2 つ目の更新は、3 文字の文字列 "101" を、ROW80_11 FID からのデータで構成されるページの行の 2 文字目のオフセットに書き込みます。例の 3 つ目の更新は、1 文字の文字列 "7" を、ROW80_11 FID からのデータで構成されるページの行の 5 文字目のオフセットに書き込みます。 このように、ページに対する更新は、非常に簡潔です。

ログ・エントリ・フォーマットの変更
ログ・エントリのデフォルト・フォーマットを、2 つの方法で変更できます。

環境変数 ESP_RMDS_SUBSCRIBE_SYMBOL_FORMAT を 1 に設定して、Event Stream Processor に流れる値を示すメッセージをデフォルトの複数行フォーマットの代わりに単一行でログ記録するように、システムを設定できます。これによって、ログ・ファイルに書き込まれたメッセージに対して特定の項目をスキャンするのが容易になります。

double のデータ型の変数の出力で表示される小数点以下の桁数を指定できます。これを行うには、-P オプションを **esp_rmds** コマンドで使用します。

デフォルトでは、Event Stream Processor に流れる値を示すログ・メッセージは、以下の例で示されているように、複数行フォーマットで書き込まれます。

```
(38079.526) @2 INFO:Publishing VOD.mGBPd 3 of 9 on stream1 as UPSERT
_ITEM_NAME_ STRING: VOD.mGBPd
_SEQUENCE_NUMBER_ INTEGER: 953
ROW80_7 STRING: off:53 size:2 value:45
```

環境変数 ESP_RMDS_SUBSCRIBE_SYMBOL_FORMAT を 1 に設定すると、これらのメッセージは、以下の例で示されているように、単一行フォーマットで書き込まれます。

```
(17.794) @5 DEBUG:stream1 p values: _ITEM_NAME_=VOD.mGBPd
_SEQUENCE_NUMBER_=2
ROW 80_3=off:78 size:2 value:20
```

-P オプションを使用して、以下の例の ask と last のように double のデータ型として宣言されている変数の表示方法を変更できます。 これは、変数の表示のみに影響し、内容には影響しません。

```
<RowDefinition id="marketfeed_RowDef">
<Column name="symbol" datatype="string" />
```

```
<Column name="service" datatype="string" />
<Column name="seq" datatype="integer" />
<Column name="stale" datatype="integer" />
<Column name="bid" datatype="money" />
<Column name="ask" datatype="double" />
<Column name="last" datatype="double" />
<Column name="volume" datatype="integer" />
<Column name="when" datatype="timestamp" />
</RowDefinition>
```

デフォルトの精度を受け入れると、double のデータ型の変数 (たとえば、以下の例の ASK) は、小数点以下 3 桁で書き込まれます。

```
(5.089) @5 INFO:Publishing EURJPY= 7 of 9 on stream1 as UPSERT
(5.090) @5 DEBUG:stream1 p values: _ITEM_NAME_=EURJPY=
_SEQUENCE_NUMBER_=1 _ITEM_STALE_=0 BID=137.4800 ASK=137.530
ACVOL_1=0
ACTIV_DATE+TIMACT=2008-10-06T21:07:00.000 (1223327220000)
```

esp_rmds コマンドの入力時にオプション -P 7 を指定すると、データ型 double の変数 (たとえば、以下の例の ASK) は、小数点以下 7 桁で書き込まれます。 他の型の変数は影響を受けません。

```
(4.913) @5 INFO:Publishing EURJPY= 7 of 9 on stream1 as UPSERT
(4.913) @5 DEBUG:stream1 p values: _ITEM_NAME_=EURJPY=
_SEQUENCE_NUMBER_=1 _ITEM_STALE_=0 BID=137.5200 ASK=137.5700000
ACVOL_1=0 ACTIV_DATE+TIMACT=2008-10-06T20:55:00.000 (1223326500000)
```

Reuters のロギング

Reuters 側の設定ファイルを使用して、Reuters のロギングを有効または無効にします。

アダプタの RMDS へのインタフェースは、ロギング機能に書き込むようにも設定できます。 Reuters 側の設定ファイル (`rfasub.cfg` と `rfapub.cfg` は、アダプタと共に提供される設定ファイル) で、ロギングを有効または無効にでき、また、ログ・ファイルのパスとファイル名を指定できます。 Reuters インタフェースも「メッセージ・ファイル」のセットをサポートします。

Reuters 側の設定ファイルには、Reuters の「ロガー」機能のための設定エントリのセットがあります。

```
¥Logger¥AppLogger¥fileLoggerEnabled                = true
¥Logger¥AppLogger¥fileLoggerFilename              = "rfasub.{p}.log"
```

この設定は、ロイター・マーケットフィード・アダプタ用の Reuters ロギングを有効にします。ログ・メッセージは、`rfasub.`**PID**`.log` ファイルに書き込まれます。ここで、**PID** はアダプタのプロセス ID です。

このセットの最初の行 `¥Logger¥AppLogger¥windowsLoggerEnabled = false` は、Windows のロギング機能に対応しています。ただし、このロギング

機能は、ロイター・マーケットフィード・アダプタに対してはサポートされません。
これらの例の行は rfasub.cfg からで、RMDS にサブスクライブするアダプタを設定します。 パブリッシュ用の設定ファイル rfapub.cfg には、同じ設定行があります (ただし、rfapub.{p}.log が **fileLoggerFilename** の値であることを除きます)。

以下に示すように、同じファイルに、コンポーネント・ロガーのための設定エントリがあります。

```
¥Logger¥ComponentLoggers¥Connections¥messageFile   ="config/
messages/RFA7_Connections.mc"
¥Logger¥ComponentLoggers¥Adapter¥messageFile       ="config/
messages/RFA7_Adapter.mc"
¥Logger¥ComponentLoggers¥SessionCore¥messageFile   ="config/
messages/RFA7_SessionLayer.mc"
¥Logger¥ComponentLoggers¥SSLED_Adapter¥messageFile ="config/
messages/RFA7_SSLED_Adapter.mc"
```

ログ・メッセージ
これらの例は、アダプタ・ログ・ファイルからの典型的なエントリを示します。

ログ・メッセージの実際のフォーマットと動作は、ログ記録されるイベントの性質と、これらのイベントに関連付けられているログ・レベルも同じく、以降のアダプタのリリースで変更される可能性があります。

- **メッセージ：** – `NOTICE:Item BARC.VX is closed: No Quality of Service is available to process subscription, timeout expired`
- **原因：** – Reuters 設定ファイルの Reuters ユーザ名の値が不正です (大文字小文字の問題)。または、マップ・ファイルの Reuters サービス名が不正です。
- **メッセージ：** – `DEBUG: Immediate flush for low latency`
- **原因：** – RMDS から受信したデータがすぐに Event Stream Processor に送信されています。
- **メッセージ：** – `NOTICE:XMLRPC ERROR-116: The connection to the server could not be established. Please make sure the server is up, and check the specified host name/port, user name/password, and encryption settings. If a host name is specified, make sure that it can be resolved through a DNS lookup. (5.092) @1 INFO:Could not connect to SP; (tigris: 12190 cimtest) will retry in 5 seconds.`
- **原因：** – Event Stream Processor を実行しているサーバに接続できません。

- **メッセージ：** – `Ignoring market data event because no significant fields updated`
- **原因：** – アダプタは Reuters からデータを受信しましたが、いずれのフィールドも Event Stream Processor ストリームの対象になりませんでした。このため、データは何も送信されませんでした。
- **メッセージ：** – `ERROR: Error publishing: PUBLICATION ERROR-442: The send method of this publication object failed.`
- **原因：** – Event Stream Processor への接続がメッセージ転送時に失敗しました。
- **メッセージ：** – `ERROR:Mismatch between platform stream (9 columns) and adapter (31 columns for stream: stream1)`
- **原因：** – アダプタで定義されているカラムの数が、ストリームのカラム数と一致しませんでした。
- **メッセージ：** – `WARNING: Event Stream Processor down, dropping all subscriptions`

  次のようなメッセージが複数回繰り返して続きます。
  `DEBUG: Unsubscribing item: EUR= service: IDN_RDF`
- **原因：** – Event Stream Processor への接続が失われました。アダプタはどこにもデータを送信できないので RMDS データへのサブスクリプションを停止します。
- **メッセージ：** –`WARNING: Discarding data rec'd after unsubscribe`
- **原因：** – アダプタがサブスクリプションを遮断する前に、さらに多くのデータが到着しました。 このデータは無視されます。Event Stream Processor への接続がないためです。
- **メッセージ：** – `DEBUG: Processing update for EUR= from service IDN_RDF`
- **原因：** – サービス "IDN_RDF" 上の RIC "EUR=" 用の更新が到着しました。
- **メッセージ：** – `WARNING: Event Stream Processor down, dropping all subscriptions`

  次のようなメッセージが複数回繰り返して続きます。
  `DEBUG: Unsubscribing item: EUR= service: IDN_RDF`
- **原因：** – Event Stream Processor への接続が失われました。 アダプタはどこにもデータを送信できないので RMDS データへのサブスクリプションを停止します。
- **メッセージ：** –`WARNING: Discarding data rec'd after unsubscribe`

- **原因：** – アダプタがサブスクリプションを遮断する前に、さらに多くのデータが到着しました。 このデータは無視されます。Event Stream Processor への接続がないためです。
- **メッセージ：** – `EMERGENCY: Fatal Error at line 0, column 0 of config file: An exception occurred! Type:RuntimeException, Message:The primary document entity could not be opened. Id=/home/sybase/adapter/trunk/src/ReutersAdapter/xxsubexample.xml`
- **原因：** – 指定された設定ファイルが利用できません。
- **メッセージ：** – `EMERGENCY: Fatal Error at line 0, column 0 of config file: An exception occurred! Type:RuntimeException, Message:The primary document entity could not be opened. Id=/home/sybase/adapter/trunk/src/ReutersAdapter/xxsubexample.xml`
- **原因：** – 指定された設定ファイルが利用できません。

ロイター OMM アダプタ

Sybase Event Stream Processor のロイター OMM アダプタは、Event Stream Processor と Reuters Market Data System (RMDS) との間のソフトウェア・インタフェースです。 このアダプタは、Reuters Open Message Model (OMM) メッセージ・フォーマットを使用します。

アダプタは、入力アダプタまたは出力アダプタとして設定できます。 入力アダプタは、RMDS 上の 1 つ以上の Reuters Instrument Code (RIC) にサブスクライブし、入力を Event Stream Processor に提供します。 出力アダプタは、Event Stream Processor からの出力を RMDS にパブリッシュします。 これによって、Event Stream Processor は Reuters のインフラストラクチャが提供するスピードと信頼性を活用して、データを配信できます。

ロイター OMM インプット・アダプタは、スキーマ検出をサポートします。 入力と出力の機能を必要とする場合は、2 つのアダプタ・インスタンスを実行します。

アダプタは、Solaris™ と Linux® のオペレーティング・システム上でのみ実行しますが、Solaris、Linux、または Windows® 上で実行する Event Stream Processor ソフトウェアと共に使用できます。

稼働条件

ロイター OMM インプット・アダプタとロイター OMM アウトプット・アダプタには、いくつかの要件があります。

入力アダプタは、次を必要とします。

- Reuters Open Message Model (OMM) プロトコルを使用する RMDS マーケット・データ接続
- 1 つ以上の金融商品についてのデータに対する、有効なサブスクリプション

出力アダプタは、次を必要とします。

- OMM プロトコルを使用してデータを RMDS に送信することをサポートする有効な接続

一般的な考慮事項

ロイター OMM アダプタを実行する各ユーザ・アカウントにユーザがアクセスできるようにし、Reuters からの入力接続と Reuters への出力接続を設定します。

ユーザ・アクセスの有効化

ロイター OMM アダプタを使用する各ユーザ・アカウントにユーザがアクセスできるようにします。

1. ユーザ・アカウントが、インストールされたソフトウェアを実行するパーミッションを持っていることを確認します。
2. 環境変数 $ESP_RMDSOMM_HOME を作成し、アダプタ配布ファイルが格納されているディレクトリのフル・パス名に設定します。
3. (オプション) 環境変数をシェル・プロファイルに追加します。
4. Event Stream Processor は、RSA、Kerberos、LDAP の認証をサポートします。インストール環境でこれらの認証方法のいずれかが使用される場合、その認証方法で動作するようにユーザ・アカウントがセットアップされていることを確認します。

Reuters からの入力接続の設定

サンプル設定ファイルを、サイトの RMDS 接続用に変更します。 複数の RMDS 接続を使用するアダプタが複数ある場合、それぞれに個別の一意な名前付き設定ファイルを使用する必要があります。 異なる名前の設定ファイルについて、入力アダプタ・マップ・ファイルのエントリを変更するか、そのファイル名を **esp_rmdsomm** コマンドの -f オプションを使用して指定します。

前提条件

- サイト固有の設定ファイルを格納するディレクトリを作成 (または選択) する。
- 環境変数 (MY_CONFIG) を作成し、そのディレクトリのフル・パス名に設定する。

手順

インストール・プロセス時に、サンプル設定ファイル (`rmdsomm.cfg`) が `$ESP_RMDSOMM_HOME/config` ディレクトリに格納されます。 このファイルは、

設定ファイルに関する Reuters フォーマットに従っており、サイト固有の情報が設定されたこのセクションがあります。

```
##
## Site-specific values for OMM Inbound - subscribing from RMDS
##
¥Connections¥Connection_RSSL¥connectionType = "RSSL"
### Caution: post value comments like below confuse RFA parsing
causing coredump
#¥Connections¥Connection_RSSL¥hostName       = "localhost" ## not
here
¥Connections¥Connection_RSSL¥hostName       = "localhost"
¥Connections¥Connection_RSSL¥rsslPort       = "14002"
¥Connections¥Connection_RSSL¥connectRetryInterval = 7000
¥Sessions¥Session1¥connectionList           = "Connection_RSSL"
```

1. 次の情報をシステム管理者から取得します。
   - 受信する RMDS OMM データの送信元のサーバの名前
   - システムが接続するマシンのポート番号
   - サブスクライブ先の Reuters サービスの名前
2. サンプル設定ファイルのコピーを `$MY_CONFIG` ディレクトリに作成します。
   ```
   cp $ESP_RMDSOMM_HOME/config/rmdsomm.cfg $MY_CONFIG
   ```
3. 通常のエディタを使用して、設定ファイルを開きます。
4. `¥Connections¥Connection_RSSL¥rsslPort` 行で、デフォルトのポート番号 (14002) を Reuters 接続で使用されるポート番号に置き換えます (異なる場合)。
5. `¥Connections¥Connection_RSSL¥hostName` 行で、tigris.sybase.com を RMDS から OMM データを受信するサーバの名前に置き換えます (囲んでいる引用符を維持します)。
   複数のサーバが RMDS からデータを受信している場合、それらの名前をすべて、優先順に基づくカンマ区切りのリストで指定します。
6. (オプション) `¥Logger¥AppLogger¥fileLoggerFilename` 行で、ログ・ファイルの名前を変更します。
   ここで指定されているデフォルトのファイル名 rfasub.{p}.log には、文字列 {p} があり、Reuters ライブラリによって、ログ・ファイルの作成時に UNIX プロセス ID に置き換えられます。
7. 変更したファイルを保存します。
   設定ファイルの他のパラメータも、ロイター OMM アダプタの機能に影響を及ぼすので、同様に変更することをおすすめします。

Reuters への出力接続の設定

サンプル設定ファイルを、サイトの RMDS 接続用に変更します。 複数の RMDS 接続を使用するアダプタが複数ある場合、それぞれに個別の一意な名前付き設定

ファイルを使用する必要があります。 異なる名前の設定ファイルについて、出力アダプタ・マップ・ファイルのエントリを変更するか、そのファイル名を **esp_rmdsomm** コマンドの -f オプションを使用して指定します。

前提条件

- サイト固有の設定ファイルを格納するディレクトリを作成 (または選択) する。
- 環境変数 (MY_CONFIG) を作成し、そのディレクトリのフル・パス名に設定する。

手順

インストール・プロセス時に、サンプル設定ファイル (`rmdsomm.cfg`) が `$ESP_RMDSOMM_HOME/config` ディレクトリに格納されます。 このファイルは、設定ファイルに関する Reuters フォーマットに従っており、非対話型と対話型で RMDS にパブリッシュするためのサイト固有の情報が設定されたセクションがあります。

1. 次の情報をシステム管理者から取得します。
   - src_dist または RMDS インフラストラクチャ・サーバがロイター OMM アダプタからの更新情報を受信するポート番号
   - Event Stream Processor から更新情報を受信するサーバの名前
2. RMDS へのパブリッシュを対話型で行うか、または非対話型で行うかを決定します。
3. Reuters からの入力接続で指定していない場合、サンプル設定ファイルのコピーを `$MY_CONFIG` ディレクトリに作成します。
   ```
   cp $ESP_RMDSOMM_HOME/config/rmdsomm.cfg $MY_CONFIG
   ```
4. エディタを使用して設定ファイルを開きます。
   a) RMDS へのパブリッシュを対話型で行う場合、対話型パブリッシュ用のサイト固有の情報セクションに移動します。 `¥Connections ¥Connection_RSSL_PROV¥connectionType` 行で、情報プロバイダ用の Reuters の用語である値 "RSSL_PROV" を参照します。
   ```
   ##
   ## Site-specific values for OMM Outbound - Interactive
   publishing to RMDS
   ##
   # Interactive publisher
   ¥Connections¥Connection_RSSL_PROV¥connectionType = "RSSL_PROV"
   ## grab a free port until the MDH is setup with 2nd src_dist
   instance
   ¥Connections¥Connection_RSSL_PROV¥rsslPort = "14007"
   ¥Connections¥Connection_RSSL_PROV¥connectRetryInterval = 7000
   ¥Connections¥Connection_RSSL_PROV¥hostName =
   ```

```
"tigris.sybase.com"
¥Sessions¥SessionOMMProv¥connectionList =
"Connection_RSSL_PROV"
```

¥Connections¥Connection_RSSL_PROV¥rsslPort 行で、デフォルトのポート番号 (14007) を、IPC サーバがロイター OMM アダプタからの更新情報を受信するポート番号に置き換えます (異なる場合)。

b) RMDS へのパブリッシュを非対話型で行う場合、非対話型パブリッシュ用のサイト固有の情報セクションに移動します。 ¥Connections ¥Connection_RSSL_CPROV¥connectionType 行で、クライアント・プロバイダ用の Reuters の用語である値 "RSSL_CPROV" を参照します。

```
##
## Site-specific values for OMM Outbound - Non-interactive
publishing to RMDS
##
# non-interactive publisher
¥Connections¥Connection_RSSL_CPROV¥connectionType =
"RSSL_CPROV"
¥Connections¥Connection_RSSL_CPROV¥hostName =
"tigris.sybase.com"
## Within Sybase, this non-standard port is a proxy to the
standard 14003
¥Connections¥Connection_RSSL_CPROV¥rsslPort = "14010"
¥Connections¥Connection_RSSL_CPROV¥connectRetryInterval = 7000
¥Sessions¥SessionOMMCProv¥connectionList =
"Connection_RSSL_CPROV"
```

¥Connections¥Connection_RSSL_CPROV¥rsslPort 行で、デフォルトのポート番号 (14010) を、IPC サーバがロイター OMM アダプタからの更新情報を受信するポート番号に置き換えます (異なる場合)。

5. ログ・ファイルの名前を変更するには、local file logging セクションに移動します。

```
##
## General values
##
## local file logging
¥Logger¥AppLogger¥windowsLoggerEnabled = false
¥Logger¥AppLogger¥fileLoggerEnabled = true
¥Logger¥AppLogger¥fileLoggerFilename = "rfa.{p}.log"
```

¥Logger¥AppLogger¥fileLoggerFilename 行で、デフォルト名 rfapub.{p}.log を使用する名前に変更します。 デフォルト・ファイル名にある文字列 {p} は、Reuters ライブラリによって、ログ・ファイルの作成時に UNIX プロセス ID に置き換えられます。

6. 変更したファイルを保存します。

入力アダプタの設定

RMDS (Reuters Market Data Service) から Event Stream Processor にデータを送信するように、入力アダプタを設定します。

必要なデータとシステムをセットアップする方法を決定してから、入力アダプタを設定します。

受信データの送信元の Event Stream Processor インスタンスについて、以下の情報を取得する必要があります。

- クラスタ環境で適用可能なセキュリティ・オプションと、ワークスペースとプロジェクトの名前。
- 使用される認証メカニズム (Kerberos、RSA、LDAP、または none)。

データの決定

受信 Reuters データをプロジェクトに取り込む方法を決定します。

また、Level1 データまたは Level2 データを必要とするかどうかを決定します。Level2 データの場合、OMM アダプタを使用します。Level1 データの場合、OMM MarketPrice メッセージまたはロイター・マーケットフィード・アダプタを使用します。

決定事項	説明
取引所	情報を取得する取引所を決定 (たとえば、NYSE、NASDAQ、Toronto など)。
RIC と FID	必要なマーケット・データを決定。 特に、アダプタが Event Stream Processor に提供する RIC (Reuters Instrument Code) と、これらの使用する金融商品タイプのためのロイター Field ID (FID) を決定します。
ストリーム	Reuters アダプタは、Event Stream Processor 上の 1 つ以上のストリームにデータを供給することが可能。 アダプタによって提供されるロイター・マーケット・データを使用するには、どの既存のデータ・ストリームをアダプタのデータ・フィードにマップするかを決定するか、または 1 つ以上の新しいストリームを定義します。

管理上の決定

プロジェクトに関して、いくつかの管理上の決定を行う必要があります。

決定事項	説明
セッション名	プロジェクトとアダプタ・マップ・ファイルを結び付ける任意の文字列。 わかりやすい文字列を使用します。 アダプタは、アダプタのインスタンスごとに 1 つのセッションのみをサポートします。

決定事項	説明
ロギングとストリーム出力用のディレクトリ	アダプタは、独自のログ・メッセージを書き込み、Reuters ログ・メッセージの個別のセットを生成可能。 設定では、これらのログ・ファイルが書き込まれるかどうかと、どこに書き込まれるかを指定します。
Sybase ユーザ・アカウント	Event Stream Processor の起動時に認証を指定していない場合を除き、アダプタが使用する、有効な Event Stream Processor ユーザ・アカウントを指定。

入力アダプタ・マップ・ファイル

マップ・ファイルは、ロイター OMM アダプタと Event Stream Processor との間のインタフェースを設定します。どのソース・ストリームが RMDS からアダプタを介してデータを受信するかを指定し、特定の RMDS Field ID (FID) をそのソース・ストリームの特定のカラムにマップします。

入力アダプタ・マップ・ファイルを作成するには、次の 3 つの主要なタスクを実行する必要があります。

- Event Stream Processor 設定ファイルで定義されている 1 つ以上のストリームのカラムに、受信データ要素を一致させる。
- アダプタからの各更新情報が、ストリームのカラム定義で定義されているように、更新対象の各ストリームの一意なキーを提供するレコードに変換できるようにする。

データ構造

データ構造には、構造上重要な 3 つの側面があります。 データ・カラム、データ型、キー値です。

- 各データ・ストリームは、1 つ以上のデータ・カラムで構成される。
- 各カラムには、データ型が設定されている。
- 各ローには、一意なキー値がある。 ソース・ストリーム定義には、「キー」カラムとして 1 つ以上のカラムが指定されています。 データは、ソース・ストリームに提供される必要があります。

受信 RMDS データ

アダプタが特定の RIC の RMDS にサブスクライブすると、RMDS は、最初に、その RIC の利用可能なすべてのマーケット・データで構成される初期イメージを送信します。 その後、RMDS は、サブスクライブしている RIC のいずれかの値が変更されると更新情報のみを送信します。

RMDS 用に定義された各 FID には、データ型が設定されています。

マーケット・データ・フィールド・マッピング
対象の Event Stream Processor ストリームの各カラムをロイター FID または「疑似フィールド」にマップします。

ストリームの各カラムに対応する FID を見つけます。 Event Stream Processor カラムのデータ型は、それをフィードするロイター FID のデータ型と互換性を持つ必要があります。

次に、互換性のある FID のデータ型と Event Stream Processor のデータ型を示します。

Event Stream Processor のデータ型	Reuters のデータ型
`integer`	`enumeration`, `time_seconds`
`integer` または `long`	`uint32`
`long`	`uint64`
`money`、`float`、`integer`、または `long`	`real32`
`money` または `float`	`real64`
`string`	`ASCII_string`, `RMTES_string`
`date` または `timestamp`	`date, time`

注意： OMM は、時間フィールドの一部としてミリ秒をサポートします。 タイムスタンプ・カラムとの間でマッピングが行われると、ミリ秒が維持されます。

Reuters Instrument Code マッピング
各受信 RMDS 更新情報の識別子は、RIC (Reuters Instrument Code) です。

RIC をストリームのデータ型 string のカラムにマップします。マップ先のストリームに適切なカラムがない場合は、ストリームにカラムを追加するか、別のストリームにマップします。

ストリームのキーとの一致
アダプタ・マップ・ファイルでは、Event Stream Processor ストリームに送信される各更新情報に、そのストリーム用に定義されている一意なキーに一致するフィールドまたはフィールドの組み合わせがあるようにアダプタを設定する必要があります。 この条件をより柔軟にするために、アダプタ設定メカニズムでは、「疑似フィールド」をサポートしています。

アダプタが RMDS から受信するマーケット・データの更新情報は、マップ・ファイルの dataField 要素または dateTimeField 要素を使用して Event Stream Processor ス

トリームのカラムにマップされます。 RMDS には、マーケット・データ以外の情報もあり、各更新情報には RIC があります。 さらに、各更新情報にシーケンス番号を付加するようにアダプタを構成できます。

これらのデータ項目をマッピング・プロセスで利用できるようにするために、マップ・ファイル・メカニズムは、「疑似フィールド」と呼ばれるこれらの要素をサポートします。

フィールド	説明	データ型
`dataField`	PRICE、SIZE などの値。	(必須。データ型は、FID とストリームのスキーマに基づいて実行時に決定される)
`dateTimeField`	日時。	`string` (オプション)
`itemName`	RIC。	`string`(必須)
`imageField`	いずれかのエントリがイメージであることを示すフラグ。	`integer` (Level2 データでは必須)
`itemStale`	項目の状態。	`integer` (オプション)
`marketByOrderKeyField`	Level 2 メッセージ用のセカンダリ・キー。	`integer` (Level 2 MARKET_BY_ORDER メッセージの場合は必須)
`marketByPriceKeyField`	Level 2 メッセージ用のセカンダリ・キー。	`integer` (Level 2 MARKET_BY_PRICE メッセージの場合は必須)
`marketMakerKeyField`	Level 2 メッセージ用のセカンダリ・キー。	`integer` (MARKET_MAKER メッセージの場合は必須)
`nullField`	null 値	(オプション、プレースホルダ)
`respTypeNumField`	メッセージのタイプを示す。	`integer` (オプション)
`sequenceNumber`	アダプタによって各受信イベントに順次に割り当てられる一意な番号 (更新を発生させるかどうかに関係なし)。	`long` (オプション)

フィールド	説明	データ型
`serviceName`	RMDS によって受信された、この RIC からのマーケット・データの送信元サービスの名前。	`string` (オプション)
`updateNumber`	アダプタによって各受信更新情報に順次に割り当てられる一意な番号。	`long` (オプション)

プロジェクトからストリーム情報の取得

Reuters ストリームについて必要な情報を収集します。

入力アダプタを設定する場合は、最初に、RMDS マーケット・データを受信する、Event Stream Processor 上のソース・ストリームを決定します。 この目的のストリームが Event Stream Processor プロジェクトにない場合は、Reuters アダプタと共に使用する新しい 1 つ以上のストリームを定義します。

ロイター OMM アダプタからデータを受信するストリームを選択 (または定義) したら、そのストリームについての情報をプロジェクト・ファイルから収集します。 Event Stream Processor プロジェクト・ファイルには、1 つ以上のストリーム定義があります。 各ストリーム定義は、Event Stream Processor の起動時にインスタンス化されるデータ・ストリームを指定します。 ストリーム定義は、以下で構成されます。

- ストリームの一意な ID
- ストリーム・データ用のデータベース・ストアと出力ファイル
- データ・ストリーム内の各ローの一意なキー値として使用されるカラムのリスト

Reuters アダプタによって提供される RMDS データを転送するストリームを決定したら、プロジェクト・ファイルのストリーム定義から情報を取得します。 プロジェクト・ファイル名に関する標準はありません。 Event Stream Processor の 2 つのインストール環境でストリーム定義が完全に異なる場合もありますが、すべてのストリーム定義にある、コンポーネントの基本セットは同じものです。

次の手順は、ロイター OMM アダプタを構成するために識別する必要のあるストリーム設定のコンポーネントを示すために、用例のプロジェクトを参照しています。

1. アダプタがデータを提供するプロジェクトを開きます。 ロイター OMM アダプタ配布には、サンプル・プロジェクト `$ESP_RMDSOMM_HOME/examples/`

`example.ccl` があり、3 つのストリーム用のスキーマ定義が設定されています。

2. ソース・ストリームの名前を見つけます。 開始 SourceStream タグは、ストリームの名前を ID 属性の値として指定します。 この例の最初のソース・ストリームの名前は、"marketByOrderStream" です。

   サブスクリプションのためにロイター OMM アダプタによって使用されるストリームは、常にソース・ストリームである必要があります。
3. キー・フィールドを決定します。 開始 SourceStream タグと終了 SourceStream タグの間の各カラム・エントリで、キー属性が "true" に設定されていることを検証します。 この例では、"marketByOrderStream" に 1 つのキー・フィールド "symbol" があります。
4. ソース・ストリーム定義のカラム・エントリの数と順序を間違わないように書き留めます。

   入力アダプタ・マップ・ファイルで、同じセットのデータを同じ順序でリストします。

入力マップ・ファイルの作成

`examples` サブディレクトリで提供されているサンプル・アダプタ・マップ・ファイルの手順を使用して、独自のアダプタ・マップ・ファイルを作成します。

1. アダプタ・マップ・ファイル用のディレクトリを選択または作成します。
2. `$ESP_RMDSOMM_HOME/examples` ディレクトリの内容をそのディレクトリにコピーします。
3. インストール環境に応じて、エディタを使用して用例ファイルを変更します。

入力アダプタの実行

ロイター OMM インプット・アダプタの設定が完了したら、実行します。

前提条件

アダプタが設定されていること。

手順

1. **esp_server** が実行しており、プロジェクトがロードされて起動されていることを確認します。
2. Event Stream Processor が RSA 認証を使用して実行している場合、次のコマンドを使用してアダプタを起動します。

```
esp_rmdsomm -a in -f mapfile -p cluster_host:cluster_port/
workspace/project ¥
 -k <private_rsa_key_file> -c username
```

3. Event Stream Processor が Kerberos/LDAP 認証を使用して実行している場合、次のコマンドを使用してアダプタを起動します。
```
esp_rmdsomm -a in -f mapfile -p cluster_host:cluster_port/
workspace/project ¥
-c username:password
```
4. Event Stream Processor が認証を必要とせずに実行している場合、次のコマンドを使用してアダプタを起動します。
```
esp_rmdsomm -a in -e -f mapfile -p cluster_host:cluster_port/
workspace/project ¥
```
5. アダプタは、最初に Event Stream Processor に接続し、次に RMDS に接続することによってサブスクリプションを起動します。 両方の接続が、すべてのデータ・フローに関して動作している必要があります。
   アダプタのログの出力先を stderr にする場合、ここに示されているように、stderr をログ・ファイルにリダイレクトすることをおすすめします (たとえば、上記のコマンド・ラインに append >& myrmdsommlog & を追加します)。

アダプタのテスト

アダプタが期待どおりに動作しない場合は、**esp_rmdsomm** コマンドを実行し、アダプタがロイター・マーケット・データを Event Stream Processor に送信していることを検証して、簡単なサニティ・チェックを行えます。

- **esp_rmdsomm** を実行。
```
esp_rmdsomm -v
```
  このコマンドは、バージョン情報を返します。 接続先の Event Stream Processor が使用するアダプタのバージョンと互換性があることを確認します。
- ロイター OMM アダプタがロイター・マーケット・データを Event Stream Processor に送信しているかどうかを簡単に確認するために、3 つの方法が用意されています。
  - スタジオまたは **esp_subscribe** コマンドを使用して、Reuters データを受信するように設定されているストリームの出力をチェック。
  - リダイレクトされたアダプタのログ・ファイル (アダプタ・マップ・ファイルで指定されます) または Reuters サブスクライバ・ログ (設定ファイル `rmdsomm.cfg` で指定されます) に対して tail コマンドを使用して、アクティビティをチェック。
  - -d7 オプションを指定して **esp_rmdsomm** コマンドを実行し、詳細な出力を生成。

複数の RIC
入力アダプタを設定する場合、複数の証券コードについて情報を入手するには複数の RIC を指定します。

これを行うには、いくつかの方法があります。

- 名前をマップ・ファイルに直接入力するか、XML ENTITY インクルード・ファイルを使用することによって、個々の RIC を指定。
- Event Stream Processor を使用して RIC のリストを指定する動的ウォッチ・リストを作成。
- 上記のオプションを組み合わせて使用。

個々の RIC
マップ・ファイルの itemList セクションに追加する各 RIC の項目要素宣言を入力します。

この例を以下に示します。

```
<itemLists service="SSL_PUB" stream="marketByOrderStream">
<itemList>
<item name="CSCO.O"/>
<item name="K.N"/>
<item name="KBN.N"/>
<item name="KBR.N"/>
<item name="ACAM.ARC"/>
<item name="IBM.ARC"/>
</itemList>
</itemLists>
```

この方法では、RIC のリストが非常に大きくなったり、リストの変更が頻繁に発生したりする場合に、リストを作成したり、保守したりすることが困難になる可能性があります。 たとえば、NYSE で取引される株式のすべての場合です。 同じストリームのすべての RIC が、同じ FID セットを使用する必要があります。 FID は取引所ごとに異なることがあるので、取引所ごとに異なる itemList と streamMap を使用します。

動的ウォッチ・リストの作成
動的ウォッチ・リストの作成手順は、多少複雑ですが、柔軟に行うこともできます。 RIC のカスタマイズされた独自のリストも指定できます。

前提条件
データを受信するための定義済みソース・ストリーム (MyInfoStream) と、手動で編集した RIC のリストが必要です。

手順

この手法も動的です。 以下の手順を使用して設定されたストリーム上で挿入または削除が発生すると、該当する RIC への RMDS サブスクリプションが起動するか、停止します。

1. サブスクライブ先の RIC のリストをアダプタにパブリッシュする、Event Stream Processor 上のストリーム (たとえば、MyListStream) を定義します。 このストリームでは、次のカラムが必要です。

カラム	説明
symbol	アダプタがサブスクライブする RIC 記号チッカ (たとえば、CSCO.O) を指定。
service	その RIC のデータを取得するためにサブスクライブする RMDS サービスを指定。
stream	アダプタがこの RIC のデータをパブリッシュするストリーム (たとえば、MyInfoStream) の名前を指定。

   4 つ目のカラム rfaQueue を指定することもできます (オプション)。
2. 最初のストリームによって要求されるデータを受信する、Event Stream Processor 上の 2 つ目のストリーム (たとえば、MyInfoStream) を定義します。
3. マップ・ファイルを編集して、サブスクリプションを追加します。

```
<subscriptions>
<subscription name="subscription1" flags="BASE" >
<stream name="MyListStream" >
<name column="3" /> <!-- symbol -->
<field column="1" name="service"/>
<field column="2" name="stream"/>
</stream>
</subscription>
</subscriptions>
```

4. 必要とする RIC のセットを指定し、それらにサブスクライブするために作成した最初のストリーム (たとえば、MyListStream) にそれらを送信します。
   a) ストリームが期待するのと同じ 6 つのカラムをカンマ区切り値 (CSV) フォーマットで持つファイルを作成します。カラムは以下のとおりです。 受信しているデータの送信元ストリーム、opcode (例の p は UPSERT を表します)、サービス、証券コード、データの送信先ストリーム。
   たとえば、エディタを使用して新しいファイル (`RIClist.csv`) を開き、これらの行を挿入します。

```
MyListStream,p,,IDN_RDF,MyInfoStream,CSCO.O
MyListStream,p,,IDN_RDF,MyInfoStream,K.N
MyListStream,p,,IDN_RDF,MyInfoStream,KBN.N
```

```
MyListStream,p,,IDN_RDF,MyInfoStream,KBN.R
MyListStream,p,,IDN_RDF,MyInfoStream,ACAM.ARC
MyListStream,p,,IDN_RDF,MyInfoStream,IBM.ARC
```

b) **esp_convert** コマンドと **esp_upload** コマンドを使用してこれらのファイルからデータを Event Stream Processor に送信します。次の例は、すべての Sybase コマンド・ライン・ツールがデフォルトのディレクトリにインストールされており、それらのディレクトリが PATH 変数に追加されていると想定しています。 そうでない場合は、この例で示されている各コマンドの前に、該当するパスを付加してください。

たとえば、ローカル・サーバのポート 11180 で実行している Event Stream Processor に、上記の手順で作成したファイルを送信するには、次のコマンドを入力します。

```
cat RIClist.csv | esp_convert -c user:password -d "," ¥
-p localhost:11180/ws1/p1 | esp_upload -c user:password -p
localhost:11180/ws1/p1
```

c) アダプタを起動します。

```
esp_rmdsomm -f mapfile -d7 -c user:password ¥
-p localhost:11180/ws1/p1 >& logfile &
```

アダプタと Event Stream Processor が別のマシンに存在する場合は、上記のコマンドで -p の後に、localhost の代わりにリモート・ホストの名前を入力します。

パフォーマンス・チューニング

入力アダプタのパフォーマンスを微調整するために使用できる、4 つの属性があります。

属性	説明
`flushInterval`	データの累積時に待機する時間間隔をマイクロ秒単位 (たとえば、5000 マイクロ秒 = 5 ミリ秒) で指定。 この間隔の終了時に、累積されたすべてのイベントが Event Stream Processor に送信されます。 送信イベントの間隔を長くすると、多くのイベントがメッセージに格納され、通信オーバヘッドが削減されます。ゼロ以外の値を flushInterval に指定すると、イベントが時間を基にして累積されます。

属性	説明
`maxRecordsPerBlock`	Event Stream Processor にアダプタが 1 回に送信する、累積されたイベントの最大数を指定。 累積されたイベントの数がこの値よりも大きいと、エンベロープまたはトランザクションが、指定した値以下のイベントで構成されるフラグメントに分割されます。 たとえば、累積されたイベントの数が 1024 (この数では、Event Stream Processor Gateway のインバウンド・キューがすぐにいっぱいになります) を超えることが予想される場合、maxRecordsPerBlock を 500 などの値に設定して、いっぱいになるのを防ぎます。
`pendingLimit`	イベントを累積する数のしきい値を指定。この値になると、累積されたイベントが Event Stream Processor に送信されます。 このパラメータをゼロに設定すると、イベントが発生するとすぐにアダプタはそのイベントをパブリッシュします (遅延が最も小さくなります)。ただし、ネットワークのオーバヘッドが非常に高くなります (更新情報ごとに 1 つの TCP/IP パケット)。 このパラメータを大きな値に設定すると、アダプタは多くのイベントが累積されるまで待機し、TCP/IP パケットに効率よく格納して、Event Stream Processor に送信します。 これによって、通信処理の負荷は削減されますが、アダプタと Event Stream Processor の両方で遅延が増加します。

属性	説明
sendAsTransactions	このパラメータは、イベントがエンベロープまたはトランザクションのいずれで送信されるかを制御。 このパラメータは、ストリームごとに指定できます。 このパラメータを true に設定すると、Event Stream Processor は、更新のグループを 1 つのトランザクションとして扱います。 トランザクションは、通常、アプリケーションレベルのワークロード削減をもたらします。これは、Event Stream Processor が、1 つのトランザクション内で同じ値に対する複数の更新を 1 つの更新にまとめるためです。 また、トランザクションに削除がある場合、削除の前の更新が廃棄されるので、さらなるワークロード削減が達成されます。 このパラメータを false に設定し、低遅延モード (pendingLimit と flushInterval の両方をゼロに設定) ではない場合、maxRecordsPerBlock を使用してエンベロープのサイズを制御します。 上記で説明されているように、通信オーバヘッドが削減されますが、トランザクション上の削減は得られません。 これは、マーケット・データ整合アプリケーションなど、各イベントをすべて必要とするアプリケーションで推奨される設定です。 一般的なルールとして、最新の更新情報のみが重要な相場値ベースのアプリケーションでは、トランザクションの設定がはるかに効率的です。 総量を計算するために各イベントをすべて処理する必要がある取引では、エンベロープを使用する必要があります。

flushInterval と pendingLimit の両方が使用されている場合、すべてのイベントが flushInterval の時間を待たずに送信され、pendingLimit (またはそれ以上) のイベントが到着すると、それらはすぐに送信されます。 アダプタは、flushInterval で指定された時間が経過するまで待機し、イベントが累積された場合には、それらを送信します。 アダプタが以前のイベントを送信しているときに pendingLimit の値以上のイベントが累積された場合、アダプタは新しいイベントをすぐに送信します (flushInterval の時間が経過するまで待機することはありません)。 アダプタがイベントを送信しているときに pendingLimit の値よりも少ないイベントが累積されている場合、アダプタは flushInterval の時間が経過するまで待機します。

rfaQueue 属性も itemLists、itemList、または item の要素のレベルで使用できます。この属性を指定すると、名前付き rfaQueue 上で Reuters から要素がサブスクライブされます。 各 rfaQueue は Reuters アダプタ内の独自のスレッドによって処理されま

す。 要求を複数のスレッドに分散することで、CPU の使用率を向上させ、遅延を削減し、全体的なアダプタのスループットを改善できます。

すべてのイメージと更新は同じキュー上で Reuters から到着するので、到着の順序の整合性は、個々の RIC すべてで維持されます。 要素のいずれにも rfaQueue を指定しない場合、すべての RIC に対して 1 つのデフォルト・キュー ("defaultQueue") が使用されます。

出力アダプタ設定

Event Stream Processor から RMDS にデータをプッシュするように出力アダプタを設定します。

提供するデータとシステムをセットアップする方法を決定してから、出力アダプタを設定します。

受信データの送信元の Event Stream Processor インスタンスについて、以下の情報を取得する必要があります。

- クラスタ環境で適用可能なセキュリティ・オプションと、ワークスペースとプロジェクトの名前。
- 使用される認証メカニズム (Kerberos、RSA、LDAP、または none)。

データの決定

Event Stream Processor に流入するどのカラムからデータをパブリッシュするかを識別します。

ロイター OMM アダプタは、ストリームからのカラムを任意の順序に並び替えることができます。 その出力は定数を構成でき、パブリッシュされた出力は複数のストリームからの値も構成できます。

ロイター OMM アウトプットの出力を計画する場合、次の点に注意する必要があります。

- データのパブリッシュ元の各ストリームに対して、出力アダプタ・マップ・ファイルに一意なキーを指定できる必要がある。 このアダプタはデータを RMDS に送信するので、一意なキーは RIC である必要があります。 MarketPrice データの場合、キーは RIC だけにできます。 Level 2 データの場合、キーには他のフィールドもある必要があります。 RIC に加えて、MarketbyPrice は PRICE と SIDE を必要とし、MarketbyOrder は ORDER_ID を必要とします。
- いずれかのストリームからパブリッシュされる各データ・カラムは、一意な FID にマップされる必要がある。
- 1 つのカラムからのデータは、パブリッシュされる出力内で繰り返し出現できる。これによって、日時値を個別の日付と時刻の値としてパブリッシュできます。

- 作業しているストリームが複数のサービスから同じ FID についてのデータを受信する場合、これらのデータ項目をサービスごとに区別し、データをサービスごとに個別に転送するようにアダプタを設定できる。
- ロイター OMM アダプタは RMDS に初めてパブリッシュするときに、設定されているすべてのカラムの値をパブリッシュする。その初期イメージの後、これらのカラムの更新が発生するごとに、アダプタは個々のカラムの更新情報のみをパブリッシュします。

Event Stream Processor のカラムのデータ型は、それをフィードするロイター FID のデータ型と互換性がある必要があります。 次の表は、一致可能な Event Stream Processor と FID のデータ型の組み合わせを示します。

Event Stream Processor のデータ型	Reuters のデータ型
`integer`	`enumeration`、`time_seconds`、`uint32`、`uint64`、または `real32` です。
`long`	`int64` または `uint64`
`money, float`	`real32` または `real64`
`string`	`ASCII_string`, `RMTES_string`
`date, timestamp`	`date`, `time`

管理上の決定
プロジェクトに関して、いくつかの管理上の決定を行う必要があります。

決定事項	説明
セッション名	プロジェクトとアダプタ・マップ・ファイルを結び付ける任意の文字列。 わかりやすい文字列を使用します。
ロギングとストリーム出力用のディレクトリ	アダプタは、独自のログ・メッセージを書き込み、Reuters ログ・メッセージの個別のセットを生成可能。 設定では、これらのログ・ファイルが書き込まれるかどうかと、どこに書き込まれるかを指定します。
Sybase ユーザ・アカウント	Event Stream Processor の起動時に認証を指定していない場合を除き、アダプタが使用する、有効な Event Stream Processor ユーザ・アカウントを指定。

Reuters 情報

ロイター OMM アダプタが RMDS にパブリッシュできるようにするには、Reuters から提供されているいくつかの情報を使用する必要があります。

- アダプタがデータを転送する Reuters サービスの名前
- RMDS によって使用される、有効な RIC (Reuters Instrument Code) と Field ID (FID) の最新のリスト
- Reuters によって割り当てられる Product Permission Code

アダプタは Reuters データ・アクセス制御システム (DACS) をサポートしていないので、Product Permission Code が、RMDS 上で転送する情報へのアクセスを許可するためのメカニズムを提供するために必要です。

FID のリスト $ESP_RMDSOMM_HOME/config/RDMFieldDictionary が、Reuters アダプタ配布の一部として提供されています。 最新のリストと他の情報については、Reuters のテクニカル・サポートから取得してください。

プロジェクトからストリーム情報の取得

プロジェクトから必要な情報を収集します。

出力アダプタを設定する最初の手順では、Event Stream Processor に流入するどのデータ要素がパブリッシュされるかを決定します。 Reuters アダプタを介して RMDS 上でパブリッシュされる項目で構成されるプロジェクトを選択 (または定義) したら、RMDS に送信するデータの取得元となるストリームから情報を収集します。

各ストリーム定義は、Event Stream Processor の起動時にインスタンス化されるデータ・ストリームを指定します。 ストリーム定義は、以下の機能を提供します。

- ストリームの一意な ID を指定する。
- データ・ストリームの各ローの一意なキーとして使用されるカラムを識別する。

Reuters アダプタによって RMDS に送信される情報を提供するストリームを決定したら、プロジェクト・ファイルのストリーム定義から情報を取得します。 プロジェクト・ファイル名に関する標準はありません。 Event Stream Processor の 2 つのインストール環境でストリーム定義が完全に異なる場合もありますが、すべてのストリーム定義にある、コンポーネントの基本セットは同じものです。

1. アダプタがデータを入手するプロジェクトを開きます。 ロイター OMM アダプタ配布には、`$ESP_RMDSOMM_HOME/examples/example.ccl` にサンプル・プロジェクトがあります。
2. プロジェクトで定義されている各ストリームの定義を使用して、以下を実行します。

a) そのストリームの開始タグの ID 属性からストリームの名前を取得します。
b) RIC を含むカラムに対してキー属性が "true" に設定されていることを検証し、そのカラムを書き留めます。 この例では、"marketByOrderStream" が、キー・フィールドとして識別される "symbol" という名前のカラムに RIC を持ちます。
c) アダプタが RMDS に送信するデータを決定します (存在する場合)。

3. RMDS に送信するデータで構成されるストリームと、ストリーム定義のどこにそれがあるかを、間違わないように書き留めます。
   出力アダプタ・マップ・ファイルで、パブリッシュする各カラムを参照します。

出力マップ・ファイルの作成
アダプタ・マップ・ファイルを作成して、出力アダプタと Event Stream Processor の間のインタフェースを設定します。

examples サブディレクトリにサンプル・アダプタ・マップ・ファイルがあります。

1. アダプタ・マップ・ファイル用のディレクトリを選択または作成します。
2. `$ESP_RMDSOMM_HOME/examples` ディレクトリの内容をそのディレクトリにコピーします。
3. インストール環境に応じて、エディタを使用して用例ファイルを変更します。

出力アダプタの実行
アダプタの設定が完了したら、実行します。

前提条件
アダプタが設定されていること。

手順

1. **esp_server** が実行しており、プロジェクトがロードされて起動されていることを確認します。
2. 次のコマンドを使用して、アダプタを起動します。
   ```
   esp_rmdsomm -a out -f mapfile -p cluster_host:cluster_port/
   workspace/project
   ```
   コマンドの実際の使用方法は、Event Stream Processor の起動方法によって異なります。 アダプタは、互換性のあるオプションを使用して起動する必要があります。 例示されているコマンド文字列には、暗号化も認証も指定されていません。いずれか、または両方を指定できます。

注意： アダプタのログの出力先を stderr にする場合、ここに示されているように、stderr をログ・ファイルにリダイレクトすることをおすすめします (たとえば、上記のコマンド・ラインに **>& myrmdsommlog &** を追加します)。

アダプタのテスト

アダプタが期待どおりに動作しない場合は、**esp_rmdsomm** コマンドを実行し、アダプタがロイター・マーケット・データを Event Stream Processor に送信していることを検証して、簡単なサニティ・チェックを行えます。

- **esp_rmdsomm** を実行。

```
esp_rmdsomm -v
```

- このコマンドは、アダプタのリリース番号とソース・ツリーのリビジョン番号をアンダースコア文字で区切って返す。 使用しているアダプタと Event Stream Processor のバージョンに互換性があることを確認します。
- ロイター OMM アダプタが RMDS にパブリッシュしていることを確認する方法が、いくつか用意されている。
  - コンソール出力のリダイレクト先のアダプタ・ログ・ファイル、または任意の Reuters パブリッシャのログ・ファイル (`rmdsomm.cfg` で指定されています) に対して tail コマンドを使用して、アクティビティを検索する。
  - **esp_subscribe** コマンドを使用してアウトバウンド・ストリームを調査し、値が変化していることを検証する。
  - RMDS ツールを使用して、出力アダプタによって提供されている RIC にサブスクライブする。
  - 入力アダプタを使用し、RMDS Market Data Hub (MDH) 経由で出力アダプタにサブスクライブする。

パフォーマンス・チューニング

複数のスレッドを使用することによって、出力アダプタのパフォーマンスを向上できます。

出力アダプタ・マップ・ファイルの subscriptions セクションには、複数のサブスクリプションを指定できます。 各サブスクリプションは個別のスレッド上でインスタンス化されるので、複数の subscription セクションを指定することによって複数スレッドでの実行がもたらすパフォーマンス向上を達成できます。

アダプタ・マップ・ファイルの分割

入力アダプタ・マップ・ファイルまたは出力アダプタ・マップ・ファイルの一部を別のファイルに書き込むことによってもパフォーマンスが向上します。

たとえば、サブスクリプション設定をマップ・ファイルに維持し、アダプタがサブスクライブする RIC のリストを別ファイルで維持します。

`$ESP_RMDSOMM_HOME/examples` のサンプル・ファイルは、この方法によって再使用が簡単にできるようになることを示しています。

`pubexample.omm.map.xml` マップ・ファイルは、3 つの「マップ・フラグメント」ファイルを参照しています。`mbo.s.mf.xml`、`mbp.s.mf.xml`, 、`mp.s.mf.xml` です。 `mbo.s.mf.xml` は、他の 3 つのマップ・ファイルからも参照されています。

マップ・ファイル・フラグメントは、XML エンティティ・メカニズムを使用してアダプタ・マップ・ファイルを構築するための再使用可能な XML ブロックです。ファイル名は、`description.parent_element.mf.xml` の形式に従います。現在の一部の記述を以下に示します。

- **mbo –** MarketByOrder
- **mbp –** MarketByPrice
- **mp –** MarketPrice

現在の parent_element は以下のとおりです。

- **sd –** Event Stream Processor のモデル・ストリーム定義
- **sms –** サブスクライバの streamMaps セクション
- **rfa –** 共通設定セクション
- **sm –** サブスクライバの streamMap
- **il –** サブスクライバの itemList
- **s –** パブリッシャのストリーム

このため、`mbo.sm.mf.xml` ファイルは、サブスクライバ・マップ・フラグメントで、MARKET_BY_ORDER メッセージの streamMap 要素で構成されることが明らかにわかります。

従属マップ・ファイルの作成

従属マップ・ファイルを作成して、マップ・ファイル設定の一部を保持します。

1. マップ・ファイルが存在するディレクトリに移動します。
2. 拡張子 `.xml` を持つ新しいファイルを作成します。

   XML バージョンの宣言を追加する必要はありません。
3. 選択した内容をマップ・ファイルから新しいファイルに挿入します。

   追加する内容は、個別に格納すると決定した、マップ・ファイルの部分により異なります。
4. (オプション) コメントを新しいファイルに追加します。
5. 編集が完了したら、ファイルを保存します。

メイン・マップ・ファイルの変更
従属ファイルを参照するように、メイン・マップ・ファイルを変更します。

1. メイン・マップ・ファイルの先頭行が以下であることを確認します。
   ```
   <?xml version="1.0"?>
   ```
2. 次の行を、XML バージョンの宣言と開始 adapter タグの間に追加します。
   ```
   <!DOCTYPE adapter SYSTEM "adapter.dtd" [
   ]>
   ```
3. 各従属マップ・ファイルについて、以下を実行します。
   a) 追加したばかりの 2 行の間に以下のようなエントリを追加します。
      ```
      <!ENTITY SUBREF SYSTEM "SUBFILE">
      ```
      ここで、SUBREF は従属ファイルを参照するために使用される文字列で、SUBFILE は従属ファイル自体のパスとファイル名です。 ファイル名とパスを引用符で囲みます。
   b) 従属マップ・ファイルに挿入した内容を削除します。
   c) 以下のような文字列を挿入して、従属マップ・ファイルからの内容をインクルードします。
      ```
      &SUBREF;
      ```
      ここで、SUBREF は従属ファイルを参照するように指定した文字列です。

コマンドの使用

esp_ommsample

esp_ommsample ユーティリティは、Reuters Market Data System (RMDS) から受信したデータを stdout に表示します。

シノプシス

```
esp_ommsample -u username [ OPTION ...]
```

説明

esp_ommsample ユーティリティは、OMM メッセージ用の RMDS からのデータ・シンクとして機能します。 このユーティリティを使用すると、ロイター OMM アダプタとモデルをセットアップすることなく、配信されるフィールドとそれらの値を確認できます。

このプロセスは、設定をコマンド・ラインから取得して、stdout に出力します。このプロセスに対して、実行する期間を指定できます。また、[Ctrl] キーを押しながら [C] を押して停止できます。

必須の引数

- **-u username –** RMDS [ENTER_VALID_USERNAME] に接続するためのユーザ名を指定。

オプション

- **-a FID_dictionary –** デフォルトの辞書の代わりに使用する辞書を指定 (`./config/RDMFieldDictionary`)。
- **-A applicationId –** デフォルト (256) を上書きする ApplicationId を指定。
- **-c Reuters config file –** Reuters 設定ファイルのパスとファイル名を指定。 通常、このファイルはロイター OMM アダプタと共有されるので、これはデフォルトで `./config/rmdsomm.cfg` に設定されます。
- **-e enum_defs –** デフォルト (`./config/enumtype.def`) を上書きするファイル名を指定。
- **-f format –** 更新メッセージのフォーマット (0、1、2、または 3) を指定。 デフォルトは 0 で、各値が個別の行にある複数行フォーマットです。 すべての値を 1 行で取得するには、1 を指定します。次に例を示します。

  ```
  207 TRIN.O|TRDPRC_1=1.14|BID=1.13|ASK=1.17|ACVOL_1=1000|
  ASK_TIME=10:26:2|
  ```

  RIC (TRIN.O) の前には、ミリ秒単位のタイムスタンプがあり、後には、"|" で区切られて FID=(値ペア) が続きます。 フィールド名と共に FID 番号を取得するには、2 を指定します。 field[FID]=(値)。 tersest、FID=(値) のフォーマットで取得するには、3 を指定します。

  区切り文字は、環境変数を使用して上書きできます。
  ESP_OMMSAMPLE_PAIR_SEPARATOR のデフォルトは '=' です。
  ESP_OMMSAMPLE_FIELD_SEPARATOR のデフォルトは '|' です。
  ESP_OMMSAMPLE_TIMESTAMP_SEPARATOR のデフォルトは ' ' です。
- **-h –** ヘルプ・メッセージを表示して終了。
- **-I instanceId –** デフォルト (1) を上書きする InstanceId を指定。
- **-m type –** 使用するメッセージ・タイプ (MMT) を、以下の中から、指定。

  ```
    1 = MarketPrice
    m = MarketMaker
    o = MarketByOrder
  [ p = MarketByPrice ]
    s = SymbolList
  ```
- **-p period –** 更新の受信を待機する秒数を指定。この時間が経過すると、終了します。デフォルトは、120 秒です。
- **-P position –** デフォルトを上書きする (yourIP/net) ポジションを指定。

- **-r service –** RMDS サービスを指定。 サイトで有効なサービス名を指定します。この値のデフォルトは、Reuters のテスト・ラボで利用できるサービス DF_EAP_LAB1 です。
- **-s symbol [symbol ...] –** 1 つ以上のサブスクライブ先証券コード (RIC) を指定。スペース区切りのリストは、シェルから保護するために、引用符で囲む必要があります。
- **-s file –** 1 つ以上のサブスクライブ先証券コード (RIC) で構成されたファイルを指定。 これらは、-s オプションで指定されたものに追加されます。
- **-v –** バージョン番号を表示して終了。

例
一般的な起動方法は次のとおりです。

```
cd $ESP_RMDSOMM_HOME/bin
./esp_ommsample -u myUsername -r MY_SERVICE -m 1 -s GOOG.O >&
esp_ommsample.out &
```

esp_rmdsomm
ロイター OMM アダプタは、RMDS からのデータを Event Stream Processor のデータに、およびその逆方向に適用します。

シノプシス

```
esp_rmdsomm -f mapFile -p host:port/workspace/project [ OPTION ...]
```

説明
esp_rmdsomm コマンドは、アダプタを、Event Stream Processor から Reuters Market Data System (RMDS) へ、またはその逆方向のデータ・ソースまたはシンクのいずれかとして起動できます。 RMDS からのサブスクライブと RMDS へのデータのパブリッシュの両方を行うには、RMDS OMM アダプタの 2 つの異なるインスタンスを実行する必要があります。

接続を記述するメタデータには、マップ・ファイル、設定ファイルなどのいくつかのパートがあります。また、Event Stream Processor の実行中のインスタンスに常駐する設定ストリームのパートがあることがあります。

ロイター OMM には、いくつかの「ドメイン」があります。 現在、MARKET_PRICE、MARKET_BY_PRICE、MARKET_BY_ORDER のみが完全にサポートされています。 MARKET_MAKER はインバウンドのみでサポートされています。 メッセージ・ドメインで想定される FID など詳細については、Reuters のマニュアルを参照してください。

プロセスは、デーモンとして実行し、設定をマップ・ファイルから取得します。SIGHUP を処理するので、`kill -s SIGHUP pid` (Linux の場合) または `kill -s HUP pid` (Solaris の場合) を入力して、アダプタをシャットダウンできます。ここ

で、pid は **esp_rmdsomm** デーモンのプロセス ID で、**ps** コマンドを使用して取得できます。 HUP シグナルではなく KILL シグナルを使用することによって、システム・リソースの完全なクリーン・アップを防止できます。

アダプタがインストールされているディレクトリの下には、doc、examples、config の 3 つのディレクトリがあり、追加情報が格納されています。 doc ディレクトリには、さまざまな設定オプションを記述する Reuters README ファイルがあります。 examples ディレクトリには、多くの機能を示す、いくつかの用例マップ・ファイルがあります。 config ディレクトリには、RMDS 設定ファイルの例があります。 少なくとも RMDS 設定ファイルを、サイト固有の情報を使用して変更する必要があります。 一般的に、マップ・ファイルも Event Stream Processor に一致するように変更します。

必須の引数

- **-f mapFile –** マーケット・データと RMDS との間をマップするために必要なメタデータが構成されているマップ・ファイルを指定。
- **-p hostname:port/workspace/project –** サーバ (クラスタ・マネージャ) に接続するための URI を指定。 たとえば、-p localhost:19011/default/prj1 は、localhost のポート 19011 を使用する ESP クラスタ・サーバのデフォルトのワークスペース内のプロジェクト prj1 を指定します。

オプション

- **-a in|out|interactive –** RMDS OMM アダプタのインスタンスがデータを Event Stream Processor に渡しているか、それから渡されたデータを受信しているかを指定。 指定できる値は、`in`、`out`、`interactive` です。 デフォルト値は `in` なので、マーケット・データにサブスクライブするときには、このオプションは、通常、省略されます。

  下位互換性のため、"subscribe" (in) と "publish" (out) は依然として許可されますが、非推奨です。
- **-c user[:password] –** 認証方法を使用しており、クレデンシャル (Kerberos、PAM、または RSA) を必要とする場合、このオプションはそれらの認証クレデンシャルを Event Stream Processor に渡します。 Event Stream Processor がこれらのクレデンシャルを正常に認証すると、接続が維持されます。それ以外の場合、Event Stream Processor は接続をすぐに閉じます。
- **-d debugLevel –** デバッグ・レベルを設定。有効な範囲は 0 ～ 7 です。0 が最小で、7 が詳細です。 デフォルトは、4 に設定されます。
- **-e –** Event Stream Processor とのすべての接続に、暗号化された OpenSSL ソケットでネゴシエート。このオプションを使用するときには、Event Stream Processor を暗号化モードで起動する必要があります。

- **-F configFile –** RMDS 設定ファイルを指定し、マップ・ファイルで指定されている設定ファイルを上書き。
- **-g gatewayHost –** Event Stream Processor ゲートウェイ・ホストを明示的に指定。
- **-G –** Kerberos 認証を使用。このオプションは、-V gssapi オプションを使用して Event Stream Processor を起動したときに必要です。
- **-h –** このコマンドの構文を説明する簡易なヘルプ・メッセージを表示。
- **-k privateRsaKeyFile –** パスワード認証の代わりに、RSA プライベート・キー・ファイル・メカニズムを使用して認証を実行。 privateRSAKeyFile には、RSA プライベート・キーの絶対パス・ファイル名を指定する必要があります。 このオプションを有効にした場合、-c オプションを使用してユーザ名を指定する必要がありますが、パスワードは必要ありません。 さらに、Event Stream Processor が -k オプションを使用して起動されている必要があります。
- **-l 0|1|2|3 –** ログ・メッセージの送信先を指定。ログ・メッセージを送信しない場合は、`0` を使用します。stderr のみに送信する場合は `1` (デフォルト)、syslog のみに送信する場合は、`2`、stderr と syslog の両方に送信する場合は、`3` を指定します。
- **-r resubscribeInterval –** RIC に再サブスクライブするまでの待機時間を秒単位で指定 (デフォルトは、300) (RIC へのサブスクリプションが CLOSED または CLOSEDRECOVER とマークされている場合、データを流れさせるためにその RIC に再サブスクライブする必要があります)。 再サブスクライブの試行を無効にするには、値として `0` を指定します。 定期的に再サブスクライブすることによって、ソースがサブスクライバにとって利用できないなどの一時的な状態から回復できます。 再サブスクライブ試行が失敗するごとに失敗イベントが生成されます。これによって、ステータスが更新されて、項目が失効しているとマークされます。
- **-s streamName –** 検出モードで実行時に使用されるストリームを指定。 このオプションは、コネクタ起動メカニズムによって使用され、マップされたカラムが検出された単一ストリームを指定します。
- **-v –** RMDS OMM アダプタのバージョンを印刷して終了。
- **-w retrySeconds –** Event Stream Processor への接続を再試行するまでの待機時間を秒単位で指定。 デフォルトは 5 です。 `0` の指定は、接続を 1 回だけ試行し、その後、再試行しないことを意味します。
- **-x optName –** その他の設定を指定。`-x help` を使用すると、指定可能な値のリストが表示されます。
- **-z publishCount –** 終了するまでに、Event Stream Processor に渡す値の数を指定。デフォルトは 0 で、終了しないことを意味します。
- **-Z subscribeCount –** 終了するまでに、RMDS に渡す値の数を指定。 デフォルトは 0 で、終了しないことを意味します。

例
コマンドを入力したマシンで、ポート 1099 を使用し、`myMap.xml` マップ・ファイルを使用してワークスペース work02 で実行中のプロジェクト proj1 を使用してロイター OMM インプット・アダプタを起動するには、次のコマンドを使用します。

```
esp_rmdsomm -c user:passwd -f myMap.xml -p localhost:1099/work02/
proj1 -d 7 &> omm.in.log &
```

loki という名前のホストで、2010 を使用し、`myMap.xml` マップ・ファイルを使用してワークスペース work01 で実行中の proj3 を使用してロイター OMM アウトプット・アダプタを起動するには、次のコマンドを使用します。

```
esp_rmdsomm -a out -c user:passwd -f myMap.xml -p loki:2010/work01/
proj3 -d 7 &> omm.out.log &
```

環境変数
ロイター OMM アダプタは、環境変数を使用して動作を指定します。

環境変数	使用箇所	説明
ESP_ACCUMULATOR_DELAY	入力	エキスパート： Event Stream Processor への接続を遅延する時間 (秒単位)。
ESP_DISABLE_REPORT_ENCODING_NULL	出力	ブランクから null への変換についての警告を停止 (bool 値で指定。デフォルトは false)。
ESP_FLUSH_INTERVAL	入力	パブリッシュの flushInterval を上書き (マイクロ秒単位)。
ESP_INTRASUBSCRIBE_DELAY	入力	Map 属性を上書き (ミリ秒単位)。
ESP_LOG_CONFIG_EVENTS	両方	設定イベント処理用のログ・レベル (1 ～ 7) を設定 (デフォルトは -1)。
ESP_MAX_RECORDS_PER_BLOCK	入力	パブリッシュの maxRecordsPerBlock を上書き (カウント数)。

環境変数	使用箇所	説明
ESP_PENDING_LIMIT	入力	パブリッシュの pendingLimit を上書き。
ESP_RETRY_INTERVAL	両方	パブリッシュの retryInterval を上書き。
ESP_RMDSOMM_DISPATCH	両方	エキスパート： N ミリ秒ごとに RFA をディスパッチ (デフォルトは、10,000)。
ESP_RMDSOMM_EVENT_TRACE	両方	エキスパート： N イベントごとに RFA イベント・トレーシングを有効化 (整数)。
ESP_RMDSOMM_HOME	両方	インストール・ディレクトリを指定。
ESP_RMDSOMM_PUBLISH_DEBUG_LEVEL	出力	値を確認するため 7 に設定 (-opt 内ではない)
ESP_RMDSOMM_PUBLISH_DEBUG_SYMBOLS	出力	記号を表示しないデフォルトの動作が上書きされたときに使用される記号のスペース区切りのリスト。設定されていない場合、すべての記号が使用されます。
ESP_RMDSOMM_SUBSCRIBE_DEBUG_LEVEL	入力	値を確認するため 7 に設定 (-opt 内ではない)
ESP_RMDSOMM_SUBSCRIBE_DEBUG_SYMBOLS	入力	上記用の記号のスペース区切りのリスト。 設定されていない場合、すべての記号が使用されます。
ESP_RMDSOMM_SUBSCRIBE_SYMBOL_FORMAT	入力	記号のリスト・フォーマットを指定。 複数行に対しては 0、単一行に対しては 1 を指定します。

環境変数	使用箇所	説明
ESP_SEND_AS_TRANSACTIONS	入力	Map 属性を上書き。
ESP_SHOW_FIELD_INFO	入力	FID、カラム、spColumn、ストリーム名を表示 (デフォルトは false)。
ESP_SHOW_SP_EVENT_DATA	出力	Event Stream Processor からのイベントのログ・レベル (1 ～ 7) を設定 (デフォルトは、-1)。

入力アダプタ・マップ・ファイル

ロイター OMM インプット・アダプタ用のマップ・ファイルの構造を以下に示します。

```
adapter                                    (required, limit one)
  |----publication                         (required, limit one)
  |----streamMaps                          (required, limit one)
  |      '----streamMap                    (required)
  |             |----dataField             (required)
  |             |----hiResTimestampField   (optional)
  |             |----imageField            (required for L2 data)
  |             |----itemName              (required, limit one)
  |             |----itemStale             (optional)
  |             |----marketByOrderKeyField (required)
  |             |----marketByPriceKeyField (required)
  |             |----marketMakerKeyField   (required)
  |             |----nullField             (optional)
  |             |----respTypeNumField      (optional)
  |             |----sequenceNumber        (optional)
  |             |----serviceName           (optional)
  |             '----updateNumber          (optional)
  |----rfa                                 (required, limit one)
  '----itemLists                           (required, limit one)
         '----itemList                     (required)
                '----item                  (optional)
```

adapter

adapter 要素は、マップ・ファイルのルート要素です。

まとめ

```
adapter                                    (required, limit one)
  |----publication                         (required, limit one)
  |----streamMaps                          (required, limit one)
```

```
  |      '----streamMap                        (required)
  |             |----dataField                 (required)
  |             |----hiResTimestampField       (optional)
  |             |----imageField                (required for L2 data)
  |             |----itemName                  (required, limit one)
  |             |----itemStale                 (optional)
  |             |----marketByOrderKeyField     (required)
  |             |----marketByPriceKeyField     (required)
  |             |----marketMakerKeyField       (required)
  |             |----nullField                 (optional)
  |             |----respTypeNumField          (optional)
  |             |----sequenceNumber            (optional)
  |             |----serviceName               (optional)
  |             '----updateNumber              (optional)
  |----rfa                                     (required, limit one)
  '----itemLists                               (required, limit one)
         '----itemList                         (required)
                '----item                      (optional)
```

親
なし

子
以下の子要素が、アダプタに対して定義されます。 これらの要素すべてが存在している必要があり、指定された順序で配置する必要があります。

名前	稼働条件
publication	1 つだけ必要
streamMaps	1 つだけ必要
rfa	1 つだけ必要
itemLists	1 つだけ必要

属性

名前	説明	稼働条件
name	ログ・エントリでこのアダプタを一意に識別する文字列	オプション

注意
なし

例
マップの各コンポーネントで提供されている例を参照してください。

dataField
streamMap 定義で、dataField 要素は、ロイター FID をソース・ストリームの 1 つのカラムにマップします。

まとめ

```
adapter                                    (required, limit one)
  |----publication                         (required, limit one)
  |----streamMaps                          (required, limit one)
  |       '----streamMap                   (required)
  |              |----dataField            (required)
  |              |----hiResTimestampField  (optional)
  |              |----imageField           (required for L2 data)
  |              |----itemName             (required, limit one)
  |              |----itemStale            (optional)
  |              |----marketByOrderKeyField (required)
  |              |----marketByPriceKeyField (required)
  |              |----marketMakerKeyField  (required)
  |              |----nullField            (optional)
  |              |----respTypeNumField     (optional)
  |              |----sequenceNumber       (optional)
  |              |----serviceName          (optional)
  |              '----updateNumber         (optional)
  |----rfa                                 (required, limit one)
  '----itemLists                           (required, limit one)
         '----itemList                     (required)
                '----item                  (optional)
```

親
streamMap

子
なし

属性

名前	説明	稼働条件
name	ソース・ストリームのこのカラムに出現するデータ項目を識別するロイター FID	必須
key	true または false (このカラムがソース・ストリームの一意なキーの一部であるかどうかによって決定)	「注意」を参照

注意
入力アダプタ・マップ・ファイルの **streamMap** セクションの各要素は、ターゲット・ソース・ストリームの RowDefinition の 1 つのカラムを表している必要があります (**streamMap** 要素の順序は、ソース・ストリームのカラムの順序と一致してい

る必要があります)。 ソース・ストリームのカラムがデータ項目 (Bid、Ask など) の場合、対応する **streamMap** エントリは、name 属性が特定の FID を識別する **dataField** 要素である必要があります。 その FID が設定されている更新を RMDS がパブリッシュするごとに、アダプタは、それを対応するカラムの値として Event Stream Processor のソース・ストリームに送信します。

値を true に設定するには、key 属性を使用する必要があります。 このカラムがストリームのキーの一部ではない場合、この属性は省略できます。

アダプタは、Event Stream Processor のスキーマを使用します。

例

```
<streamMap name="marketByOrder">
        <itemName key="true"/>
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <dataField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

上記の例は、marketByOrder ストリームのカラム 4 ～ 8 をロイター FID の BID、ASK、TRDPRC_1、ACVOL_1 にマップします。

dateTimeField

streamMap で、**dateTimeField** 要素は、Reuters の日付または時刻の FID (またはいずれか) を Event Stream Processor のソース・ストリームの日付、時刻、またはその両方のカラムにマップします。

まとめ

```
adapter                                    (required, limit one)
  |----publication                         (required, limit one)
  |----streamMaps                          (required, limit one)
  |      '----streamMap                    (required)
  |             |----dataField             (required)
  |             |----hiResTimestampField   (optional)
  |             |----imageField            (required for L2 data)
  |             |----itemName              (required, limit one)
  |             |----itemStale             (optional)
  |             |----marketByOrderKeyField (required)
  |             |----marketByPriceKeyField (required)
  |             |----marketMakerKeyField   (required)
  |             |----nullField             (optional)
  |             |----respTypeNumField      (optional)
  |             |----sequenceNumber        (optional)
  |             |----serviceName           (optional)
```

```
  |             '----updateNumber            (optional)
  |----rfa                                    (required, limit one)
  '----itemLists                              (required, limit one)
         '----itemList                        (required)
                '----item                     (optional)
```

親
streamMap

子
なし

属性

名前	説明	稼働条件
dateName	RMDS によって提供される日付値の FID	「注意」を参照
timeName	RMDS によって提供される時刻値の FID	「注意」を参照

注意
Event Stream Processor データ・ストリームの日付／時刻情報のデータ型として最も一般に使用されるのは、日付と時刻の両方を組み合わせた dateTime データ型です。 ただし、ほとんどの場合、RMDS によって提供され、ロイター OMM アダプタによって Event Stream Processor に送信される更新には、日付と時刻が別の FID として構成されます。

マップ・ファイルは、**dateTimeField** 要素を提供します。この要素は、この矛盾を修復するために、日付と時刻に対して個別の属性を提供します。 これによって、2 つの FID (1 つは日付、1 つは時刻) をソース・ストリーム定義の同じカラムにマップできます。

これら 3 つの属性のいずれかを必ず使用してください。 dateTime を使用する場合、単独で使用する必要があります。 dateName 属性と timeName 属性は、個別に、または一緒に使用できます。

各 FID の値は、Reuters 側の設定ファイルで参照されている FID リストにリストされているものと一致する必要があります (アダプタと共に提供されている FID リストの名前は `appendix_a` です)。 このファイルは、設定ファイル `rmdsomm.cfg` で参照されます。

例

```
<streamMap name="marketByOrderStream">
        <itemName key="true"/>
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale/>
```

```
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <dataField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

この例は、TIMACT と ACTIV_DATE の FID を組み合わせて、Event Stream Processor のソース・ストリーム marketByOrderStream の 9 番目のカラムにマップします。

hiResTimestampField

hiResTimestampField 要素は、通常のタイムスタンプを高精度のタイムスタンプに置き換えます。

まとめ

```
adapter                                        (required, limit one)
  |----publication                             (required, limit one)
  |----streamMaps                              (required, limit one)
  |      '----streamMap                        (required)
  |             |----dataField                 (required)
  |             |----hiResTimestampField       (optional)
  |             |----imageField                (required for L2 data)
  |             |----itemName                  (required, limit one)
  |             |----itemStale                 (optional)
  |             |----marketByOrderKeyField     (required)
  |             |----marketByPriceKeyField     (required)
  |             |----marketMakerKeyField       (required)
  |             |----nullField                 (optional)
  |             |----respTypeNumField          (optional)
  |             |----sequenceNumber            (optional)
  |             |----serviceName               (optional)
  |             '----updateNumber              (optional)
  |----rfa                                     (required, limit one)
  '----itemLists                               (required, limit one)
         '----itemList                         (required)
                '----item                      (optional)
```

親

streamMap

子

なし

属性

名前	説明	稼働条件
name	長整数の相対タイムスタンプ	必須

注意
この要素は、Solaris オペレーティング・システムを実行しているマシン上でのみ使用できます。

例
```
<streamMap name="marketByOrderStream">
        <itemName key="true"/>
        <!-- serviceName / -->
        <hiResTimestampField name="TIME"/>
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <dataField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

imageField

imageField 要素は、Event Stream Processor のローが最初のスナップショット・イメージの一部であるかどうかを示します。

まとめ
```
adapter                                     (required, limit one)
  |----publication                          (required, limit one)
  |----streamMaps                           (required, limit one)
  |       '----streamMap                    (required)
  |              |----dataField             (required)
  |              |----hiResTimestampField   (optional)
  |              |----imageField            (required for L2 data)
  |              |----itemName              (required, limit one)
  |              |----itemStale             (optional)
  |              |----marketByOrderKeyField (required)
  |              |----marketByPriceKeyField (required)
  |              |----marketMakerKeyField   (required)
  |              |----nullField             (optional)
  |              |----respTypeNumField      (optional)
  |              |----sequenceNumber        (optional)
  |              |----serviceName           (optional)
  |              '----updateNumber          (optional)
  |----rfa                                  (required, limit one)
  '----itemLists                            (required, limit one)
         '----itemList                      (required)
                '----item                   (optional)
```

親
streamMap

子
なし

属性

名前	説明	稼働条件
name	整数 (イメージの一部の場合は 1、そうでない場合は 0)	Level 2 で必須

注意
プロジェクトは、スナップショット・イメージのすべてのローを 1 つのトランザクションとして処理する必要があります。

例

```
        <name column="0"/>
        <keyField column="1" name="mbpkey" />
        <imageField column="2" />
<!-- summary fields -->
        <stale column="4" />
        <field column="5" name="CURRENCY" />
        <field column="6" name="ACTIV_DATE" />
        <field column="7" name="PROD_PERM" />
<!-- end summary fields -->
```

item
item 要素は、ロイター OMM アダプタのサブスクライブ先の RIC を識別します。

まとめ

```
adapter                                     (required, limit one)
  |----publication                          (required, limit one)
  |----streamMaps                           (required, limit one)
  |      '----streamMap                     (required)
  |             |----dataField              (required)
  |             |----hiResTimestampField    (optional)
  |             |----imageField             (required for L2 data)
  |             |----itemName               (required, limit one)
  |             |----itemStale              (optional)
  |             |----marketByOrderKeyField  (required)
  |             |----marketByPriceKeyField  (required)
  |             |----marketMakerKeyField    (required)
  |             |----nullField              (optional)
  |             |----respTypeNumField       (optional)
  |             |----sequenceNumber         (optional)
  |             |----serviceName            (optional)
  |             '----updateNumber           (optional)
  |----rfa                                  (required, limit one)
  '----itemLists                            (required, limit one)
         '----itemList                      (required)
                '----item                   (optional)
```

親
itemList

子
なし

属性

名前	説明	稼働条件
name	アダプタがサブスクライブする RIC。	必須
rfaQueue	指定する場合、デフォルトの rfaQueue の名前に置き換わる rfaQueue の名前。この名前によって、個別のスレッドがこのキューに対して使用されます。	オプション
service	RMDS を介して受信データを提供する Reuters Service の名前。	既に親の itemList 要素または itemLists 要素で指定している場合はオプション、それ以外は必須
stream	この RIC の更新が Event Stream Processor に渡されるソース・ストリーム。	既に親の itemList 要素または itemLists 要素で指定している場合はオプション、それ以外は必須

注意
name 属性の値は、サービス上の有効な RIC である必要があります。

ストリーム名をここに指定した場合、この RIC の更新は、そのストリームで Event Stream Processor に渡されます。 ここでストリームを指定しない場合、**itemList** レベルで指定したストリームが使用されます。

指定するストリームは、マップ・ファイル内の別の場所で定義されている **streamMap** に一致する必要があります。

例

```
<itemLists service="SSL_PUB" stream="marketByOrderStream">
        <itemList service="IDN_RDF" >
                <item name="EUR=" />
                <item name="EURJPY=" stream="stream6" />
        </itemList>
</itemLists>
```

これらの 2 つの **item** 要素は、アダプタを RIC の EUR= と EURJPY= にサブスクライブします。 EUR= の更新は、**itemLists** 要素で設定されているストリーム marketByOrderStream に送信されます。 EURJPY= の更新は、ストリーム stream6 に

送信されます。これは、**item** レベルのストリーム属性が、**itemLists** レベルの属性を上書きするためです。

itemList
itemList 要素には、**item** 要素の 1 つ以上のインスタンスがあります。

まとめ

```
adapter                                          (required, limit one)
  |----publication                               (required, limit one)
  |----streamMaps                                (required, limit one)
  |      '----streamMap                          (required)
  |             |----dataField                   (required)
  |             |----hiResTimestampField         (optional)
  |             |----imageField                  (required for L2 data)
  |             |----itemName                    (required, limit one)
  |             |----itemStale                   (optional)
  |             |----marketByOrderKeyField       (required)
  |             |----marketByPriceKeyField       (required)
  |             |----marketMakerKeyField         (required)
  |             |----nullField                   (optional)
  |             |----respTypeNumField            (optional)
  |             |----sequenceNumber              (optional)
  |             |----serviceName                 (optional)
  |             '----updateNumber                (optional)
  |----rfa                                       (required, limit one)
  '----itemLists                                 (required, limit one)
         '----itemList                           (required)
                '----item                        (optional)
```

親
itemLists

子

名前	稼働条件
item	0 個以上が必要

属性

名前	説明	稼働条件
rfaQueue	指定する場合、デフォルトの rfaQueue の名前に置き換わる rfaQueue の名前。この名前によって、個別のスレッドがこのキューに対して使用されます。	オプション

名前	説明	稼働条件
service	RMDS を介して受信データを提供する Reuters Service の名前。	既に親の itemList 要素またはすべての子 item 要素で指定している場合はオプション、それ以外は必須
stream	このリストの項目に指定した RIC で更新を受信する Event Stream Processor のソース・ストリームの名前。	既に親の itemList 要素またはすべての子 item 要素で指定している場合はオプション、それ以外は必須

注意
この要素のストリーム名を指定してアダプタを設定し、このセクションにある項目ごとの更新をそのストリームにプッシュします (この指定は項目レベルで上書きできます)。

アダプタでは、itemLists の下に複数の itemList 要素を設定できます。このため、アダプタの 1 つのインスタンスが複数の RIC グループからの更新を異なる Event Stream Processor のソース・ストリームに送信するように設定できます。

指定するストリームは、マップ・ファイルの他の場所で streamMap の name 属性の値を使用して定義された streamMap の 1 つと一致している必要があります。

rfaQueue 属性を使用することによって、スケーラビリティを制御します。

例
```
<itemLists service="SSL_PUB" stream="marketByOrderStream">
        <itemList service="IDN_RDF" >
        <item name="EUR=" />
        <item name="EURJPY=" stream="stream6" />
        </itemList>
</itemLists>
```

この **itemList** 要素は service 属性を IDN_RDF に設定し、親 **itemLists** 要素で定義されている SSL_PUB service 属性を上書きします。

itemLists
itemLists 要素には、**itemList** 要素の 1 つ以上のインスタンスがあります。

まとめ
```
adapter                                         (required, limit one)
  |----publication                              (required, limit one)
  |----streamMaps                               (required, limit one)
  |      '----streamMap                         (required)
  |             |----dataField                  (required)
  |             |----hiResTimestampField        (optional)
  |             |----imageField                 (required for L2 data)
  |             |----itemName                   (required, limit one)
```

```
  |             |----itemStale                (optional)
  |             |----marketByOrderKeyField   (required)
  |             |----marketByPriceKeyField   (required)
  |             |----marketMakerKeyField     (required)
  |             |----nullField               (optional)
  |             |----respTypeNumField        (optional)
  |             |----sequenceNumber          (optional)
  |             |----serviceName             (optional)
  |             '----updateNumber            (optional)
  |----rfa                                   (required, limit one)
  '----itemLists                             (required, limit one)
         '----itemList                       (required)
                '----item                    (optional)
```

親
adapter

子

名前	稼働条件
itemList	1 つは必須で、複数もサポート

属性

名前	説明	稼働条件
name	アダプタに関連したログ・エントリに表示される文字列。	オプション
rfaQueue	指定する場合、デフォルトの rfaQueue の名前に置き換わる rfaQueue の名前。この名前によって、個別のスレッドがこのキューに対して使用されます (item レベルで上書き可能なデフォルト)。	オプション
service	Reuters サービスの名前で、RMDS を通して入力データを提供する (item レベルで上書き可能なデフォルト)。	すべての子 item 要素が指定または継承するように、子 itemLists 要素か item 要素またはその両方が指定されている場合はオプションで、それ以外は必須
stream	このセクションの item リストで指定されている RIC で更新を受信する Event Stream Processor のソース・ストリームの名前。デフォルトは item レベルで上書きできます。	すべての子 item 要素が指定または継承するように、子 itemLists 要素か item 要素またはその両方が指定されている場合はオプションで、それ以外は必須

注意
このセクションの各 **itemList** インスタンスは、アダプタがサブスクライブする 1 つ以上の RIC のリストです。

サービスの値を Reuters 管理者から取得します。

例
```
<itemLists service="SSL_PUB" stream="marketByOrderStream">
        <itemList service="IDN_RDF" >
                <item name="EUR=" />
                <item name="EURJPY=" stream="stream6" />
        </itemList>
</itemLists>
```

itemLists 要素は、service 属性を SSL_PUB に、stream 属性を marketByOrderStream に設定します。 これらの属性は、**itemList** レベルか **item** レベル、またはその両方で継承または上書きされます。

itemName
streamMap 定義で、**itemName** 要素が、RMDS 更新から RIC を送る Event Stream Processor のソース・ストリームのローを識別します。

まとめ
```
adapter                                     (required, limit one)
  |----publication                          (required, limit one)
  |----streamMaps                           (required, limit one)
  |       '----streamMap                    (required)
  |              |----dataField             (required)
  |              |----hiResTimestampField   (optional)
  |              |----imageField            (required for L2 data)
  |              |----itemName              (required, limit one)
  |              |----itemStale             (optional)
  |              |----marketByOrderKeyField (required)
  |              |----marketByPriceKeyField (required)
  |              |----marketMakerKeyField   (required)
  |              |----nullField             (optional)
  |              |----respTypeNumField      (optional)
  |              |----sequenceNumber        (optional)
  |              |----serviceName           (optional)
  |              '----updateNumber          (optional)
  |----rfa                                  (required, limit one)
  '----itemLists                            (required, limit one)
         '----itemList                      (required)
                '----item                   (optional)
```

親
streamMap

子
なし

属性

名前	説明	稼働条件
key	true または false (このカラムがソース・ストリームの一意なキーの一部であるかどうかによって決定)	最初の「注意」を参照

注意
key 属性を使用する必要はありません。 下位互換性があります。

itemName 要素は、RIC または証券コードを伝達するソース・ストリームのカラムに対応するために、streamMap に挿入する必要があります。 このカラムがソース・ストリームのキーを構成する場合、**key** 属性を true に設定する必要があります。

この要素は、RMDS から直接受信するデータ・フィードの一部ではないデータ項目を指定する「疑似フィールド」の 1 つです。

例

```
<streamMap name="marketByOrderStream">
        <itemName key="true"/>
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <dataField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

この例で、ソース・ストリームの最初のカラムは、アダプタからの任意の更新の RIC 値を伝達するものとして識別されます。 ストリームのキーの一部としても識別されます。

itemStale
streamMap 定義で、**itemStale** 要素は、受信 RMDS データが失効したかどうかを示すインジケータを伝達する、Event Stream Processor のソース・ストリームのカラムを識別します。

まとめ

```
adapter                                          (required, limit one)
  |----publication                               (required, limit one)
```

```
  |----streamMaps                                (required, limit one)
  |       '----streamMap                         (required)
  |              |----dataField                  (required)
  |              |----hiResTimestampField        (optional)
  |              |----imageField                 (required for L2 data)
  |              |----itemName                   (required, limit one)
  |              |----itemStale                  (optional)
  |              |----marketByOrderKeyField      (required)
  |              |----marketByPriceKeyField      (required)
  |              |----marketMakerKeyField        (required)
  |              |----nullField                  (optional)
  |              |----respTypeNumField           (optional)
  |              |----sequenceNumber             (optional)
  |              |----serviceName                (optional)
  |              '----updateNumber               (optional)
  |----rfa                                       (required, limit one)
  '----itemLists                                 (required, limit one)
         '----itemList                           (required)
                '----item                        (optional)
```

親
streamMap

子
なし

属性

名前	説明	稼働条件
name	アダプタに関連したログ・エントリに表示される文字列	オプション

注意
ソース・ストリームのカラムのいずれかが「失効」カラムの場合は、この要素を **streamMap** で使用します。

RMDS 自体は「失効」フラグを通常のマーケット・データに付加しませんが、RMDS を介してサブスクライブしている別のサービスによって付加される場合には、そのまま送信します。 この要素が **streamMap** で使用されていると、アダプタは、RMDS から「失効」フラグを受信した場合にゼロ以外の更新値を送信します。または、RMDS からのデータの受信を停止します。失効理由は、3 つのビットを使用して示されます。

アダプタ・ステータス・ビット	説明
0	unknown = 初期状態
1	connectionInLoss
2	connectionOutLoss

アダプタ・ステータス・ビット	説明
3 - 7	予約済み

データ・ステータス・ビット	説明
8	疑わしいデータ
9	未指定 (初期化中)
10 - 15	予約済み

ストリーム・ステータス・ビット	説明
16	不明 = 初期化中
17	nonStreaming = スナップショットのみとして設定
18	closedRecover = ストリームは閉じられたが、再試行可能
19	closed = ストリームは閉じられ、復旧しない
20	redirected = フェールオーバ状態の一部
21	stale = OMM 用
22 - 24	予約済み

例

```
<streamMap name="marketByOrderStream">
        <itemName key="true"/>
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <dataField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

この例では、ソース・ストリームの 4 番目のカラムは、アダプタが「失効」通知を受信するか、RMDS からのデータ受信を停止した場合に、更新されるカラムとして識別されます。

marketByOrderKeyField

marketByOrderKeyField 要素は、MARKET_BY_ORDER ドメインのメッセージのセカンダリ・キーです。

まとめ

```
adapter                                    (required, limit one)
  |----publication                         (required, limit one)
  |----streamMaps                          (required, limit one)
  |       '----streamMap                   (required)
  |              |----dataField            (required)
  |              |----hiResTimestampField  (optional)
  |              |----imageField           (required for L2 data)
  |              |----itemName             (required, limit one)
  |              |----itemStale            (optional)
  |              |----marketByOrderKeyField (required)
  |              |----marketByPriceKeyField (required)
  |              |----marketMakerKeyField  (required)
  |              |----nullField            (optional)
  |              |----respTypeNumField     (optional)
  |              |----sequenceNumber       (optional)
  |              |----serviceName          (optional)
  |              '----updateNumber         (optional)
  |----rfa                                 (required, limit one)
  '----itemLists                           (required, limit one)
         '----itemList                     (required)
                '----item                  (optional)
```

親
streamMap

子
なし

属性

名前	説明	稼働条件
name	文字列	Level 2 で必須

注意
通常、ORDER_ID FID はセカンダリ・キーとして指定されます。

例

```
<streamMaps>
    <streamMap name="MarketByOrderStream"
messageType="MARKET_BY_ORDER">
        &marketByOrder;
    </streamMap>
```

```
    <streamMap name="MarketByPriceStream"
messageType="MARKET_BY_PRICE">
        &marketByPrice;
    </streamMap>
    <streamMap name="MarketMakerStream" messageType="MARKET_MAKER">
        &marketMaker;
    </streamMap>
</streamMaps>
```

marketByPriceKeyField

marketByPriceKeyField 要素は、MARKET_BY_PRICE ドメインのメッセージのセカンダリ・キーです。

まとめ

```
adapter                                     (required, limit one)
  |----publication                          (required, limit one)
  |----streamMaps                           (required, limit one)
  |       '----streamMap                    (required)
  |              |----dataField             (required)
  |              |----hiResTimestampField   (optional)
  |              |----imageField            (required for L2 data)
  |              |----itemName              (required, limit one)
  |              |----itemStale             (optional)
  |              |----marketByOrderKeyField (required)
  |              |----marketByPriceKeyField (required)
  |              |----marketMakerKeyField   (required)
  |              |----nullField             (optional)
  |              |----respTypeNumField      (optional)
  |              |----sequenceNumber        (optional)
  |              |----serviceName           (optional)
  |              '----updateNumber          (optional)
  |----rfa                                  (required, limit one)
  '----itemLists                            (required, limit one)
         '----itemList                      (required)
                '----item                   (optional)
```

親

streamMap

子

なし

属性

名前	説明	稼働条件
name	文字列としての PRICE + SIDE	Level 2 で必須

注意
この要素は、Event Stream Processor で解析されません。 同じ RIC のオーダーブック・ローを維持するためのセカンダリ・キーとしてのみ使用されます。

例
```
<streamMaps>
    <streamMap name="MarketByOrderStream"
messageType="MARKET_BY_ORDER">
        &marketByOrder;
    </streamMap>
    <streamMap name="MarketByPriceStream"
messageType="MARKET_BY_PRICE">
        &marketByPrice;
    </streamMap>
    <streamMap name="MarketMakerStream" messageType="MARKET_MAKER">
        &marketMaker;
    </streamMap>
</streamMaps>
```

marketMakerKeyField
marketMakerKeyField 要素は、MARKET_MAKER ドメインのメッセージのセカンダリ・キーです。

まとめ
```
adapter                                     (required, limit one)
  |----publication                          (required, limit one)
  |----streamMaps                           (required, limit one)
  |      '----streamMap                     (required)
  |             |----dataField              (required)
  |             |----hiResTimestampField    (optional)
  |             |----imageField             (required for L2 data)
  |             |----itemName               (required, limit one)
  |             |----itemStale              (optional)
  |             |----marketByOrderKeyField  (required)
  |             |----marketByPriceKeyField  (required)
  |             |----marketMakerKeyField    (required)
  |             |----nullField              (optional)
  |             |----respTypeNumField       (optional)
  |             |----sequenceNumber         (optional)
  |             |----serviceName            (optional)
  |             '----updateNumber           (optional)
  |----rfa                                  (required, limit one)
  '----itemLists                            (required, limit one)
         '----itemList                      (required)
                '----item                   (optional)
```

親
streamMap

子
なし

属性

名前	説明	稼働条件
name	文字列、通常 MMID	Level 2 で必須

注意
なし

例

```
<streamMaps>
    <streamMap name="MarketByOrderStream"
messageType="MARKET_BY_ORDER">
        &marketByOrder;
    </streamMap>
    <streamMap name="MarketByPriceStream"
messageType="MARKET_BY_PRICE">
        &marketByPrice;
    </streamMap>
    <streamMap name="MarketMakerStream" messageType="MARKET_MAKER">
        &marketMaker;
    </streamMap>
</streamMaps>
```

nullField

streamMap で、**nullField** 要素は、常に null 値を Event Stream Processor のソース・ストリームに配信するプレースホルダとして機能します。 これによって、必要な設定を行うために、ソース・ストリームに余分なフィールドを追加できます。

まとめ

```
adapter                                     (required, limit one)
  |----publication                          (required, limit one)
  |----streamMaps                           (required, limit one)
  |      '----streamMap                     (required)
  |             |----dataField              (required)
  |             |----hiResTimestampField    (optional)
  |             |----imageField             (required for L2 data)
  |             |----itemName               (required, limit one)
  |             |----itemStale              (optional)
  |             |----marketByOrderKeyField  (required)
  |             |----marketByPriceKeyField  (required)
  |             |----marketMakerKeyField    (required)
  |             |----nullField              (optional)
  |             |----respTypeNumField       (optional)
  |             |----sequenceNumber         (optional)
  |             |----serviceName            (optional)
```

```
  |             '----updateNumber            (optional)
  |----rfa                                    (required, limit one)
  '----itemLists                              (required, limit one)
         '----itemList                        (required)
                '----item                     (optional)
```

親
streamMap

子
なし

属性

名前	説明	稼働条件
name	アダプタに関連したログ・エントリに表示される文字列	オプション
dateName	アダプタに関連したログ・エントリに表示される文字列	オプション
timeName	アダプタに関連したログ・エントリに表示される文字列	オプション

注意
プロジェクトを試行する場合、**dataField** 要素または **dateTimeField** 要素を **nullField** に置き換えて、ストリームの任意のカラムへのデータ送信を一時的に停止できます。

次に示すように、一時的に置き換える **dataField** または **dateTimeField** の属性を変更する必要はありません。

例

```
<streamMap name="marketByOrderStream">
        <itemName key="true"/>
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <nullField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

この例では、ソース・ストリームの 6 番目のカラムは、アダプタからの各更新で null 値を受信するプレースホルダとして識別されます。 これには、デバッグのために置換される **dataField** の名前があります。

publication

publication 要素は、アダプタのこのインスタンスの基本パブリッシュ情報を指定します。

まとめ

```
adapter                                     (required, limit one)
  |----publication                          (required, limit one)
  |----streamMaps                           (required, limit one)
  |      '----streamMap                     (required)
  |             |----dataField              (required)
  |             |----hiResTimestampField    (optional)
  |             |----imageField             (required for L2 data)
  |             |----itemName               (required, limit one)
  |             |----itemStale              (optional)
  |             |----marketByOrderKeyField  (required)
  |             |----marketByPriceKeyField  (required)
  |             |----marketMakerKeyField    (required)
  |             |----nullField              (optional)
  |             |----respTypeNumField       (optional)
  |             |----sequenceNumber         (optional)
  |             |----serviceName            (optional)
  |             '----updateNumber           (optional)
  |----rfa                                  (required, limit one)
  '----itemLists                            (required, limit one)
         '----itemList                      (required)
                '----item                   (optional)
```

親

adapter

子

なし

属性

名前	説明	稼働条件
flushInterval	アダプタが待機するマイクロ秒数を指定する。アダプタはこの時間が経過するまでイベントを累積し、経過すると、Event Stream Processor に送信します。 ゼロ以外の値を flushInterval に設定した場合、時間に基づいてイベントを累積します。	オプション (デフォルトは 1000)
intraSubscribeDelay	アダプタがサブスクリプション要求後に次の要求まで待機するマイクロ秒数を指定する。	オプション (デフォルトは 100)

名前	説明	稼働条件
maxRecordsPerBlock	アダプタが一度に Event Stream Processor に送信する累積イベントの最大数を指定する。 これによって、イベントが大量に累積されたときに、トランザクションまたはエンベロープのそれぞれのフラグメントのサイズが小さくなります。 たとえば、maxRecordsPerBlock の値が 50 に設定されて、140 個のイベントが累積された場合、アダプタはエンベロープまたはトランザクションを 3 つのフラグメントとして送信します。	オプション (デフォルトは 256)
name	ログ・ファイル・エントリ内でアダプタ・インスタンスを識別する文字列を指定する。	オプション
pendingLimit	アダプタが累積するイベント数を指定する。この数に達すると、アダプタは Event Stream Processor にイベントを送信します。 pendingLimit を使用すると、イベントはカウント数に基づいてイベントを累積します。	オプション (デフォルトは 256)
retryInterval	アダプタが RMDS への接続試行を待機する秒数を指定する。待機時間がこの秒数に達すると、アダプタはシャットダウンします。	オプション (デフォルトは 5)
sendAsTransactions	更新のグループを 1 つのトランザクションとして扱う場合は true に設定する。1 つのエンベロープ内の個別のローとして扱う場合は false に設定します。	オプション (デフォルトは false)

注意
アダプタのパフォーマンスを最適化できます。これを行うには、pendingLimit 属性と flushInterval 属性を、アダプタが Event Stream Processor と通信するために使用するパブリッシュ／サブスクライブのインタフェースからの maxRecordsPerBlock 属性と sendAsTransactions 属性と一緒に使用します。 詳細については、「パフォーマンス・チューニング」を参照してください。

取引所によっては、マルチパートのメッセージとして初期イメージが送信され、大きいデータ・セットが生成されることがあります。 intraSubscribeDelay 属性を使用すると、これらのサブスクリプションのペースを維持し、アダプタが初期イメージに対応できなくなるのを防止できます。 デフォルト値は 0 です。この値は、短い RIC リストに適しています。 0 以外の値に設定されると、アダプタはサブスクリプション要求後に次の要求まで設定したミリ秒間待機します。 推奨値は 10 です。

例

```
<publication name="RMDS Adapter - low latency" retryInterval="5"
        flushInterval="0" pendingLimit="0" sendAsTransactions="0" />
```

respTypeNumField

respTypeNumField 要素は、カラムに RMDS respTypeNum 値を設定します。

まとめ

```
adapter                                          (required, limit one)
  |----publication                               (required, limit one)
  |----streamMaps                                (required, limit one)
  |      '----streamMap                          (required)
  |             |----dataField                   (required)
  |             |----hiResTimestampField         (optional)
  |             |----imageField                  (required for L2 data)
  |             |----itemName                    (required, limit one)
  |             |----itemStale                   (optional)
  |             |----marketByOrderKeyField       (required)
  |             |----marketByPriceKeyField       (required)
  |             |----marketMakerKeyField         (required)
  |             |----nullField                   (optional)
  |             |----respTypeNumField            (optional)
  |             |----sequenceNumber              (optional)
  |             |----serviceName                 (optional)
  |             '----updateNumber                (optional)
  |----rfa                                       (required, limit one)
  '----itemLists                                 (required, limit one)
         '----itemList                           (required)
                '----item                        (optional)
```

親

streamMap

子

なし

属性

名前	説明	稼働条件
name	アダプタに関連したログ・エントリに使用される文字列	オプション

注意

最初のスナップショット・イメージの場合、**respTypeNumField** は、UNSOLICITED に対しては 1、SOLICITED に対しては 0 の値を持ちます。 更新では、他の値が設定されることがあります。 詳細については、RMDS のマニュアルを参照してください。

例

```
<itemName key="true" /> <!-- str: the RIC -->
<marketByPriceKeyField key="true"/> <!-- str: SIDE + PRICE as a key
-->
<imageField name="imageIn" />
<updateNumber name="upd" /> <!-- generated by Adapter -->
<respTypeNumField name="rtn" />
```

rfa

rfa 要素は、サブスクライブ・マップ・ファイルを Reuters 側の設定ファイルにリンクします。

まとめ

```
adapter                                     (required, limit one)
  |----publication                          (required, limit one)
  |----streamMaps                           (required, limit one)
  |       '----streamMap                    (required)
  |             |----dataField              (required)
  |             |----hiResTimestampField    (optional)
  |             |----imageField             (required for L2 data)
  |             |----itemName               (required, limit one)
  |             |----itemStale              (optional)
  |             |----marketByOrderKeyField  (required)
  |             |----marketByPriceKeyField  (required)
  |             |----marketMakerKeyField    (required)
  |             |----nullField              (optional)
  |             |----respTypeNumField       (optional)
  |             |----sequenceNumber         (optional)
  |             |----serviceName            (optional)
  |             '----updateNumber           (optional)
  |----rfa                                  (required, limit one)
  '----itemLists                            (required, limit one)
         '----itemList                      (required)
               '----item                    (optional)
```

親

adapter

子

なし

属性

名前	説明	稼働条件
config	サブスクリプションのための Reuters 側の設定ファイルの絶対パスとファイル名 (アダプタと共に提供されるサンプル・ファイルは $ESP_RMDSOMM_HOME/config/rmdsomm.cfg)	必須
configDatabaseName	RFA に設定する必要あり	必須
enumFile	各列挙型とその値の範囲をリストする、Reuters 提供のファイルのフル・パス名	最初の「注意」を参照
fidFile	有効な FID をすべてリストした、Reuters 提供のファイルのフル・パス名	2 番目の「注意」を参照
sessionName	サブスクリプション用に Reuters 側の設定ファイルで定義されているセッション名を参照	必須
blank	ブランク用に使用するマーカを指定	オプション
blankInt32	ブランク Int32 フィールド用に使用するマーカを指定	オプション
blankInt64	ブランク Int64 フィールド用に使用するマーカを指定	オプション
blankMoney	ブランクの通貨型フィールド用に使用するマーカを指定	オプション
blankString	ブランクの文字列フィールド用に使用するマーカを指定	オプション
blankDate	ブランクの日付フィールド用に使用するマーカを指定	オプション
blankTimestamp	ブランクのタイムスタンプ・フィールド用に使用するマーカを指定	オプション
blankDouble	ブランクの倍精度フィールド用に使用するマーカを指定	オプション

注意
デフォルトの enumFile は、`$ESP_RMDSOMM_HOME/config/enumtype.def` ですが、別のファイルも指定できます。

デフォルトの fidFile は、`$ESP_RMDSOMM_HOME/config/RDMFieldDictionary` ですが、別のファイルも指定できます。

例
```
<rfa config="$ESP_RMDSOMM_HOME/config/rmdsomm.cfg"
        sessionName="Session1" />
```

この例では、ロイター OMM アダプタは、ファイル `rmdsomm.cfg` の Reuters 側設定を指し示します。 この設定ファイルのリスト行は、次のとおりです。

```
¥Sessions¥Session1¥connectionList                          =
"Connection_SSLED"
```

この行は、設定ファイルの他の行によって参照されるセッション名を定義します。 マップ・ファイルは sessionName 属性のセッション名を参照するときに、アダプタを、その名前で識別される Reuters 側の設定パラメータにリンクします。

sequenceNumber
streamMap 定義では、**sequenceNumber** 要素が、RMDS からデータの一部として提供される一意の数ではなく、アダプタが生成する一意の数によって設定される Event Stream Processor のソース・ストリームのカラムをマップします。

まとめ
```
adapter                                        (required, limit one)
  |----publication                             (required, limit one)
  |----streamMaps                              (required, limit one)
  |       '----streamMap                       (required)
  |              |----dataField                (required)
  |              |----hiResTimestampField      (optional)
  |              |----imageField               (required for L2 data)
  |              |----itemName                 (required, limit one)
  |              |----itemStale                (optional)
  |              |----marketByOrderKeyField    (required)
  |              |----marketByPriceKeyField    (required)
  |              |----marketMakerKeyField      (required)
  |              |----nullField                (optional)
  |              |----respTypeNumField         (optional)
  |              |----sequenceNumber           (optional)
  |              |----serviceName              (optional)
  |              '----updateNumber             (optional)
  |----rfa                                     (required, limit one)
  '----itemLists                               (required, limit one)
         '----itemList                         (required)
                '----item                      (optional)
```

親
streamMap

子
なし

属性

名前	説明	稼働条件
key	true または false (このカラムがソース・ストリームの一意なキーの一部であるかどうかによって決定)	「注意」を参照
name	ログ・エントリに表示される文字列	オプション

注意
アダプタは、サブスクライブ先の RIC の個別のカウンタを維持します。 RIC の更新を受信するたびに、アダプタはその RIC のカウンタを増やします。 この数値は、**sequenceNumber** 要素でマップされるソース・ストリームのカラムに送信されます。

ソース・ストリーム定義には、以下のようなカラム指定があります。

```
<Column datatype="long" name="Id"/>
```

この行はソース・ストリームの一意な ID を指定します。 **sequenceNumber** 疑似フィールドは、入力アダプタ・マップ・ファイルにあるこのカラムに適しています。

値を true に設定するには、key 属性を使用する必要があります。 このカラムがストリームのキーの一部ではない場合、この属性は省略できます。

例

```
<streamMap name="marketByOrderStream">
        <itemName key="true"/>
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <dataField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

この例では、ソース・ストリームの 3 番目のカラムが、アダプタによって提供されるシーケンス番号にマップされています。このカラムはソース・ストリームの一意なキーの一部としても識別されます。

serviceName
streamMap 定義で、**serviceName** 要素は、Event Stream Processor のソース・ストリームのカラムを、アダプタが提供するサービス識別子にマップします。

まとめ

```
adapter                                          (required, limit one)
  |----publication                               (required, limit one)
  |----streamMaps                                (required, limit one)
  |       '----streamMap                         (required)
  |              |----dataField                  (required)
  |              |----hiResTimestampField        (optional)
  |              |----imageField                 (required for L2 data)
  |              |----itemName                   (required, limit one)
  |              |----itemStale                  (optional)
  |              |----marketByOrderKeyField      (required)
  |              |----marketByPriceKeyField      (required)
  |              |----marketMakerKeyField        (required)
  |              |----nullField                  (optional)
  |              |----respTypeNumField           (optional)
  |              |----sequenceNumber             (optional)
  |              |----serviceName                (optional)
  |              '----updateNumber               (optional)
  |----rfa                                       (required, limit one)
  '----itemLists                                 (required, limit one)
         '----itemList                           (required)
                '----item                        (optional)
```

serviceName によって提供される識別子は、サブスクライブ先の 2 つの異なるサービスによって提供される RIC の名前空間スコープを提供するために使用されることがあります。

親
streamMap

子
なし

属性

名前	説明	稼働条件
key	true または false (このカラムがソース・ストリームの一意なキーの一部であるかどうかによって決定)	「注意」を参照

注意
値を true に設定するには、key 属性を使用する必要があります。 このカラムがストリームのキーの一部ではない場合、この属性は省略できます。

例

```
<streamMap name="marketByOrderStream">
        <itemName key="true"/>
        <!-- serviceName / -->
        <sequenceNumber />
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <dataField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

この例では、コメントアウトされているため、アダプタによって提供されるサービス名にソース・ストリームのいずれのカラムもマップされません。

streamMap

インプット・マップ・ファイルの **streamMap** 要素は、Event Stream Processor のソース・ストリームのカラムとアダプタによってサブスクライブされる RMDS FID との間のマッピングを定義します。

まとめ

```
adapter                                     (required, limit one)
  |----publication                          (required, limit one)
  |----streamMaps                           (required, limit one)
  |      '----streamMap                     (required)
  |             |----dataField              (required)
  |             |----hiResTimestampField    (optional)
  |             |----imageField             (required for L2 data)
  |             |----itemName               (required, limit one)
  |             |----itemStale              (optional)
  |             |----marketByOrderKeyField  (required)
  |             |----marketByPriceKeyField  (required)
  |             |----marketMakerKeyField    (required)
  |             |----nullField              (optional)
  |             |----respTypeNumField       (optional)
  |             |----sequenceNumber         (optional)
  |             |----serviceName            (optional)
  |             '----updateNumber           (optional)
  |----rfa                                  (required, limit one)
  '----itemLists                            (required, limit one)
         '----itemList                      (required)
                '----item                   (optional)
```

親

streamMaps

子

以下の子要素は、**streamMap** に対して定義されます。 これらの子要素は、任意の順序で指定できますが、特定の **streamMap** の場合、子要素の順序は、ソース・ストリームのカラムの順序を反映している必要があります (プロジェクトで定義されている順序)。 これによって、ソース・ストリームの適切なローに RMDS 更新を配信するようにアダプタが設定されます。

名前	稼働条件
dataField	1 つは必須で、複数もサポート
dateTimeField	0 個以上をサポート
imageField	Level 2 データで必須
itemName	1 つは必須で、複数もサポート
itemStale	0 または 1 をサポート
marketByOrderKeyField	Level 2 MARKET_BY_ORDER メッセージの場合に必須
marketByPriceKeyField	Level 2 MARKET_BY_PRICE メッセージの場合に必須
marketMakerKeyField	Level 2 MARKET_MAKER メッセージの場合に必須
nullField	0 個以上をサポート
respTypeNumField	0 個以上をサポート
sequenceNumber	0 個以上をサポート
serviceName	0 個以上をサポート
updateNumber	0 個以上をサポート

属性

名前	説明	稼働条件
name	RMDS 更新のマップ先のソース・ストリームを識別し、Event Stream Processor プロジェクトで定義されているソース・ストリームの名前に一致する必要あり	必須
sendAsTransactions	更新のグループを 1 つのトランザクションとして扱う場合は true、1 つのエンベロープ内の個別のローとして扱う場合は false	オプション (デフォルトは false)

注意
なし

例
```
<streamMaps>
        <streamMap name="marketByOrderStream">
                <itemName key="true"/>
                <!-- serviceName / -->
                <sequenceNumber />
                <itemStale/>
                <dataField name="BID"/>
                <dataField name="ASK"/>
                <dataField name="TRDPRC_1"/>
                <dataField name="ACVOL_1"/>
                <dateTimeField timeName="TIMACT"
dateName="ACTIV_DATE"/>
        </streamMap>
</streamMaps>
```

この例は、アダプタの一連の更新を Event Stream Processor のソース・ストリーム marketByOrderStream にマップします。 このソース・ストリームに向けられたすべての更新は、upsert opcode を使用して追加されます。

その更新がこのソース・ストリームに送信される RIC は、marketByOrderStream を参照するマップ・ファイルの別の場所の **itemList** で指定されます。

streamMaps
入力マップ・ファイルの **streamMaps** 要素には、1 つ以上の **streamMap** 要素があります。

まとめ
```
adapter                                     (required, limit one)
  |----publication                          (required, limit one)
  |----streamMaps                           (required, limit one)
  |      '----streamMap                     (required)
  |             |----dataField              (required)
  |             |----hiResTimestampField    (optional)
  |             |----imageField             (required for L2 data)
  |             |----itemName               (required, limit one)
  |             |----itemStale              (optional)
  |             |----marketByOrderKeyField  (required)
  |             |----marketByPriceKeyField  (required)
  |             |----marketMakerKeyField    (required)
  |             |----nullField              (optional)
  |             |----respTypeNumField       (optional)
  |             |----sequenceNumber         (optional)
  |             |----serviceName            (optional)
  |             '----updateNumber           (optional)
  |----rfa                                  (required, limit one)
```

```
  '----itemLists                                (required, limit one)
         '----itemList                          (required)
                '----item                       (optional)
```

親
adapter

子

名前	稼働条件
streamMap	1 つは必須で、複数もサポート

属性
なし

注意
このセクションの各 **streamMap** インスタンスは、Reuters アダプタからの受信 FID を Event Stream Processor のソース・ストリームのカラムにマップします。

ストリームには 1 つの **streamMap** が必要です。

例

```
<streamMaps>
        <streamMap name="marketByOrderStream">
                <itemName key="true"/>
                <!-- serviceName / -->
                <sequenceNumber />
                <itemStale/>
                <dataField name="BID"/>
                <dataField name="ASK"/>
                <dataField name="TRDPRC_1"/>
                <dataField name="ACVOL_1"/>
                <dateTimeField timeName="TIMACT"
dateName="ACTIV_DATE"/>
        </streamMap>
</streamMaps>
```

updateNumber
streamMap 定義では、**updateNumber** 要素が、RMDS からデータの一部として提供される一意の数ではなく、アダプタが生成する一意の数によって設定される Event Stream Processor のソース・ストリームのカラムをマップします。

まとめ

```
adapter                                       (required, limit one)
  |----publication                            (required, limit one)
  |----streamMaps                             (required, limit one)
  |      '----streamMap                       (required)
```

```
  |             |----dataField                (required)
  |             |----hiResTimestampField      (optional)
  |             |----imageField               (required for L2 data)
  |             |----itemName                 (required, limit one)
  |             |----itemStale                (optional)
  |             |----marketByOrderKeyField    (required)
  |             |----marketByPriceKeyField    (required)
  |             |----marketMakerKeyField      (required)
  |             |----nullField                (optional)
  |             |----respTypeNumField         (optional)
  |             |----sequenceNumber           (optional)
  |             |----serviceName              (optional)
  |             '----updateNumber             (optional)
  |----rfa                                    (required, limit one)
  '----itemLists                              (required, limit one)
       '----itemList                          (required)
             '----item                        (optional)
```

親
streamMap

子
なし

属性

名前	説明	稼働条件
key	true または false (このカラムがソース・ストリームの一意なキーの一部であるかどうかによって決定)	「注意」を参照
name	ログ・エントリに表示される文字列	オプション

注意
アダプタは、カラムがストリームの一意なキーの一部であるかどうかをスキーマから推測します。key 属性は、下位互換性を維持するためにのみサポートされています。

アダプタは、サブスクライブ先の RIC の個別のカウンタを維持します。 RIC の更新を受信するたびに、アダプタはその RIC のカウンタを増やします。 この数値は、**updateNumber** 要素でマップされるストリームのカラムに送信されます。

多くのソース・ストリーム定義には、以下のようなカラム指定があります。

```
<Column datatype="int64" name="Id"/>
```

この行はソース・ストリームの一意な ID を指定します。 **updateNumber** 疑似フィールドは、入力アダプタ・マップ・ファイルにあるこのカラムに適しています。

例

```
<streamMap name="marketByOrderStream">
        <itemName key="true"/>
        <updateNumber />
        <itemStale/>
        <dataField name="BID"/>
        <dataField name="ASK"/>
        <dataField name="TRDPRC_1"/>
        <dataField name="ACVOL_1"/>
        <dateTimeField timeName="TIMACT" dateName="ACTIV_DATE"/>
</streamMap>
```

この例では、ソース・ストリームの 2 番目のカラムが、アダプタによって提供される更新番号にマップされています。このカラムはソース・ストリームの一意なキーの一部としても識別されます。 その他の例については、`$ESP_RMDSOMM_HOME/examples` ディレクトリを参照してください。

出力アダプタ・マップ・ファイルの XML 構文

ロイター OMM アウトプット・アダプタ用のマップ・ファイルの構文を以下に示します。

```
adapter                         (required, limit one)
  |----rfa                                (required, limit one)
  '----subscriptions                      (required, limit one)
         '----subscription                (required)
                '----stream               (required)
                       |----name          (required, limit one)
                       |----stale         (optional)
                       |----field         (required)
                       '----constant      (optional)
```

adapter

adapter 要素は、マップ・ファイルのルート要素です。

まとめ

```
adapter                         (required, limit one)
  |----rfa                                (required, limit one)
  '----subscriptions                      (required, limit one)
         '----subscription                (required)
                '----stream               (required)
                       |----name          (required, limit one)
                       |----stale         (optional)
                       |----field         (required)
                       '----constant      (optional)
```

すべての設定要素は、開始と終了の **adapter** タグの間にネストされる必要があります。

親
なし

子
以下の子要素が **adapter** 用に定義されます。 これらのすべての要素は、指定された順序で存在する必要があります。

名前	稼働条件
rfa	1 つだけ必要
subscriptions	1 つだけ必要

属性
なし

注意
なし

例
子要素の例を参照してください。

constant
constant 要素は、アダプタによって RMDS にパブリッシュされる定数値で構成されるデータ項目を定義します。

まとめ

```
adapter                          (required, limit one)
  |----rfa                                 (required, limit one)
  '----subscriptions                       (required, limit one)
         '----subscription                 (required)
                '----stream                (required)
                       |----name           (required, limit one)
                       |----stale          (optional)
                       |----field          (required)
                       '----constant       (optional)
```

親
stream

子
なし

属性

名前	説明	稼働条件
name	アダプタによってパブリッシュされるイメージ内のこのデータ項目に関連付けられている名前	必須
value	この constant の値 (このデータ項目が RMDS にパブリッシュされる場合は常に同じ)	必須

注意
アダプタは起動時に、完全なイメージを RMDS にパブリッシュします。このイメージには、マップ・ファイルで定義されているすべてのデータ項目があります。その後に変更が発生した場合は、Event Stream Processor が失効して復旧した場合を除いて、アダプタはデータ項目の更新された値のみをパブリッシュします。 これは、**constant** の値がパブリッシュされるのは、完全なイメージがパブリッシュされた場合のみであることを意味します。

例

```
<stream name="stream1" >
        <name column="0"/>
        <stale column="3" name="ACVOL_1"/>
        <field column="1" name="DSPLY_NAME" />
        <field column="4" name="BID" precision="47" />
        <field column="5" name="ASK" precision="0" />
        <field column="6" name="TRDPRC_1"/>
        <field column="7" name="ACVOL_1"/>
        <constant name="PROD_PERM" value="1"/>
</stream>
```

この例は、PROD_PERM と呼ばれる constant を定数値 "1" で定義します。この constant は、パブリッシュ名 subscription1 の下のストリーム stream1 からのデータ値と共にパブリッシュされます。

field
出力アダプタ・マップ・ファイルの **stream** 定義では、**field** はパブリッシュするストリーム内のカラムを指定します。

まとめ

```
adapter                         (required, limit one)
  |----rfa                                (required, limit one)
  '----subscriptions                      (required, limit one)
         '----subscription                (required)
                '----stream               (required)
                       |----name          (required, limit one)
                       |----stale         (optional)
```

```
                        |----field        (required)
                        '----constant     (optional)
```

親
stream

子
なし

属性

名前	説明	稼働条件
column	パブリッシュされるストリーム内のソース・カラムの位置を表す番号 (ストリーム内の最初のカラムの番号は、"0")	column または columnName のいずれかが必要
columnName	ストリームの一意な識別子を伝達する、Event Stream Processor ストリーム内のカラムの名前	column または columnName のいずれかが必要
name	RMDS にパブリッシュされたときにこのデータ値を識別する FID	必須
precision	パブリッシュされた値内の小数点以下の総桁数を指定する整数 (たとえば、1.23 の precision は 2)	オプション

注意
precision 属性を指定できるのは、データ型 double のカラムのみです。

例

```
<stream name="stream1" >
        <name column="0"/>
        <stale column="3" name="ACVOL_1"/>
        <field column="1" name="DSPLY_NAME" />
        <field column="4" name="BID" precision="47" />
        <field column="5" name="ASK" precision="0" />
        <field column="6" name="TRDPRC_1"/>
        <field column="7" name="ACVOL_1"/>
        <constant name="PROD_PERM" value="1"/>
</stream>
```

この例では、アダプタは、stream1 と呼ばれる Event Stream Processor ストリームの 4 番目、5 番目、6 番目、7 番目のカラムの更新情報を、それぞれ BID、ASK、TRDPRC_1、ACVOL_1 の名前のデータ項目としてパブリッシュするように設定されます。

name
出力アダプタ・マップ・ファイルの **stream** 定義では、**name** は、各更新情報を識別するために使用される値を提供する、ソース・ストリーム内のカラムを指定します。

まとめ

```
adapter                          (required, limit one)
  |----rfa                                 (required, limit one)
  '----subscriptions                       (required, limit one)
         '----subscription                 (required)
                '----stream                (required)
                       |----name           (required, limit one)
                       |----stale          (optional)
                       |----field          (required)
                       '----constant       (optional)
```

親
stream

子
なし

属性

名前	説明	稼働条件
column	ストリームの一意な識別子を伝達する、ストリーム内のカラムの位置を表す番号 (ストリーム内の最初のカラムの番号は、"0")	column または columnName のいずれか
columnName	ストリームの一意な識別子を伝達する、ストリーム内のカラムの名前	column または columnName のいずれか

注意
出力アダプタは、RMDS を単純なメッセージ・バスとして使用します。パブリッシュされる更新情報は、Reuters プロトコルに準拠する必要はありません。 これは、この要素によって指定されるカラムが Reuters RIC である必要はありませんが、Reuters RIC 構文に従う必要があることを意味します。

ソース・ストリームの一意なキーが 2 つ以上のカラムで構成されている場合は、name 要素を service 要素の 1 つ以上のインスタンスと組み合わせて、完全に一意な名前を使用して更新をパブリッシュするようにアダプタを設定できます。

例

```
<stream name="stream1" >
        <name column="0"/>
        <stale column="3" name="ACVOL_1"/>
        <field column="1" name="DSPLY_NAME" />
        <field column="4" name="BID" precision="47" />
        <field column="5" name="ASK" precision="0" />
        <field column="6" name="TRDPRC_1"/>
        <field column="7" name="ACVOL_1"/>
        <constant name="PROD_PERM" value="1"/>
</stream>
```

この例では、stream1 の最初のカラムが一意な識別子または「キー」カラムとして識別されます。

rfa

rfa 要素は、アダプタの Reuters 側を設定するための情報を提供します。これには、Reuters 側の設定ファイルへの明示的な参照があります。

まとめ

```
adapter                          (required, limit one)
  |----rfa                                 (required, limit one)
  '----subscriptions                       (required, limit one)
         '----subscription                 (required)
                '----stream                (required)
                       |----name           (required, limit one)
                       |----stale          (optional)
                       |----field          (required)
                       '----constant       (optional)
```

親

adapter

子

なし

属性

名前	説明	稼働条件
serviceName	ロイター OMM アダプタによって送信される各更新のヘッダにあるサービス名	オプション
config	パブリッシュのための Reuters 側の設定ファイルの絶対パスとファイル名 (アダプタと共に提供されるサンプル・ファイルは `$ESP_RMDSOMM_HOME/config/rmdsomm.cfg`)	必須

名前	説明	稼働条件
sessionName	パブリッシュ用に Reuters 側の設定ファイルで名前付けされ、定義されているセッションを参照	必須
configDatabaseName	Reuters データベース名への参照	オプション
blankDate	ブランクの日付フィールド用に使用するマーカ	オプション
blankDouble	ブランクの倍精度フィールド用に使用するマーカ	オプション
blankInt32	ブランクの Int32 フィールド用に使用するマーカ	オプション
blankInt64	ブランクの Int64 フィールド用に使用するマーカ	オプション
blankMoney	ブランクの通貨型フィールド用に使用するマーカ	オプション
blankString	ブランクの文字列フィールド用に使用するマーカ	オプション
blankTimestamp	ブランクのタイムスタンプ・フィールド用に使用するマーカ	オプション
enumFile	各列挙型とその値の範囲をリストする、Reuters 提供のファイルのフル・パス名	オプション (デフォルトは、enumType 定義)
fidFile	有効な FID をすべてリストした、Reuters 提供のファイルのフル・パス名	オプション (デフォルトは、RDMField 辞書)

注意
なし

例

```
<rfa serviceName="IDN_RDF"
        config="$ESP_RDMSOMM_HOME/config/rmdsomm.cfg"
        sessionName="Session1" configDatabaseName="RFA" />
```

この例では、ロイター OMM アダプタは、ファイル `rmdsomm.cfg key` の Reuters 側設定を指し示します。この設定ファイルのコメント化されていない最初の 5 行は次のとおりです。

```
¥Connections¥Connection_RSSL¥connectionType = "RSSL"
¥Connections¥Connection_RSSL¥hostName = "tigris.sybase.com"
¥Connections¥Connection_RSSL¥rsslPort = "14002"
¥Connections¥Connection_RSSL¥connectRetryInterval = 7000
¥Sessions¥Session1¥connectionList = "Connection_RSSL"
```

これらの行の最後は、マップ・ファイルで **sessionName** として定義されているセッション名を暗黙的に定義しています。 このセッション名に関する `rmdsomm.cfg` キーから他の 3 つの行。 これは、**sessionName** の値が、マップ・ファイルのこの publication セクションを `.cfg` ファイルの設定セットにどのように結びつけているかを示します。

アダプタがこの設定を使用してパブリッシュする場合、各更新情報は、**serviceName** "IDN_RDF" を使用して識別されます。

stale

出力アダプタのマップ・ファイルの **stream** 定義で、**stale** 要素は、ストリームが失効すると値が "0" から "1" に変化する、ソース・ストリームのカラムを識別します。

まとめ

```
adapter                          (required, limit one)
  |----rfa                                 (required, limit one)
  '----subscriptions                       (required, limit one)
         '----subscription                 (required)
                '----stream                (required)
                       |----name           (required, limit one)
                       |----stale          (optional)
                       |----field          (required)
                       '----constant       (optional)
```

たとえば、ストリーム内のデータ・ソースの 1 つがそれ以上更新されない場合、ストリームは「失効」に変更したと見なされます。

親

stream

子

なし

属性

名前	説明	稼働条件
column	セカンダリ・キー値を持つカラムの位置を表す番号 (ストリーム内の最初のカラムの番号は、"0")	必須
name	FID にマップ (パブリッシュ) できるように、失効カラムを識別する文字列	オプション

注意

なし

例

```
<stream name="stream1" >
        <name column="0"/>
        <stale column="3" name="ACVOL_1"/>
        <field column="1" name="DSPLY_NAME" />
        <field column="4" name="BID" precision="47" />
        <field column="5" name="ASK" precision="0" />
        <field column="6" name="TRDPRC_1"/>
        <field column="7" name="ACVOL_1"/>
        <constant name="PROD_PERM" value="1"/>
</stream>
```

この例では、stream1 の 3 番目のカラムが「失効」カラムとして識別されます。「失効」カラムが指定された場合、そのカラムの値はパブリッシュされ、RIC は失効としてマーク付けされます。

stream

出力アダプタのマップ・ファイルの [subscription] セクションで、アダプタがデータを取得するストリームを識別します。このデータは RMDS にパブリッシュされます。

まとめ

```
adapter                          (required, limit one)
  |----rfa                                 (required, limit one)
  '----subscriptions                       (required, limit one)
         '----subscription                 (required)
                '----stream                (required)
                       |----name           (required, limit one)
                       |----stale          (optional)
                       |----field          (required)
                       '----constant       (optional)
```

親

subscription

子

名前	稼働条件
name	1 つ
stale	オプション
field	1 つ以上
constant	オプション

属性

名前	説明	稼働条件
exitOnStreamExit	ブール属性。 true の場合、ストリームまたは Event Stream Processor が存在するか、接続が失われると、esp_rmdsomm は終了します。	オプション (デフォルトは false)
finalizer	この文字列は、イベントが Event Stream Processor に 1 つもパブリッシュされずに heartbeat で指定されたミリ秒が経過した場合に実行するアクションを指定する。	オプション
heartbeat	この整数は、ミリ秒単位の待機時間を指定する。イベントが Event Stream Processor に 1 つもパブリッシュされない状態がこの時間を過ぎて継続すると、finalizer アクションが実行されます。	オプション
name	アダプタがデータを受信するストリーム名。このデータは、RMDS でパブリッシュされます。	必須
platformExitOnStreamDrop	ブール属性。 true の場合、このサブスクリプションが接続を切断すると、Event Stream Processor は終了します。	オプション (デフォルトは false)
platformQueueSize	Event Stream Processor キューのバイト単位のサイズ。	オプション (デフォルトは 8000)

注意

name 属性の値は、Event Stream Processor プロジェクトで定義する必要があります。

Event Stream Processor プロジェクトのストリームはいずれも、マップ・ファイルの 1 つの **stream** セクションにのみマップされます。

例

```
<stream name="stream1">
        <name column="0"/>
        <field column="4" name="TRDPRC_1"/>
        <field column="9" name="BID" precision="5"/>
</stream>
```

この例では、名前 stream1 のストリームからデータをパブリッシュするように Event Stream Processor が設定されています。

サブスクリプション

subscription 要素には、**stream** 要素の 1 つ以上のインスタンスが含まれています。これによって、1 つ以上のストリームからデータを受信するようにアダプタを設定できます。

まとめ

```
adapter                          (required, limit one)
  |----rfa                                 (required, limit one)
  '----subscriptions                       (required, limit one)
         '----subscription                 (required)
                '----stream                (required)
                       |----name           (required, limit one)
                       |----stale          (optional)
                       |----field          (required)
                       '----constant       (optional)
```

出力アダプタのマップ・ファイルには、複数の **subscription** セクションを構成できます。 実行時に、各 **subscription** セクションに対するパブリッシュ・メカニズムが個別のスレッドにインスタンス化されます。これによって、スケーラビリティが確保されます。

親

subscriptions

子

名前	稼働条件
stream	1 つ以上

属性

名前	説明	稼働条件
name	RMDS でパブリッシュされた更新とログ・ファイル・エントリに表示されるこのサブスクリプションの名前	必須

注意

なし

例

```
<subscriptions>
    <subscription name="subscription1" >
        <stream name="stream1" >
            <name column="0"/>
            <field column="4" name="BID"/>
            <field column="5" name="ASK"/>
```

```
            <field column="6" name="TRDPRC_1"/>
            <field column="7" name="ACVOL_1"/>
            <constant name="PROD_PERM" value="1"/>
        </stream>
    </subscription>
</subscriptions>
```

この例は、Event Stream Processor 上の stream1 からのいくつかのカラムを名前 subscription1 を使用してパブリッシュするようにアダプタを設定します。

サブスクリプション

subscriptions 要素には、1 つ以上の **subscription** 要素があります。

まとめ

```
adapter                          (required, limit one)
  |----rfa                                 (required, limit one)
  '----subscriptions                       (required, limit one)
         '----subscription                 (required)
                '----stream                (required)
                       |----name           (required, limit one)
                       |----stale          (optional)
                       |----field          (required)
                       '----constant       (optional)
```

親

adapter

子

名前	稼働条件
サブスクリプション	1 つ以上

属性

なし

注意

このセクションの各 **subscription** インスタンスは、アダプタが RMDS にパブリッシュするデータの 1 つのセットを定義します。

例

個別の **subscription** インスタンスの例を参照してください。

ロギング機能

ロイター OMM アダプタは、2 つの異なるロギング・メカニズムをサポートします。

独自のロギング・メカニズムに加えて、ロイター OMM アダプタは、Reuters 側のロギングを利用できます。 これらの両方とも、アダプタのパフォーマンスをチェックし、問題を診断するために使用できます。

これらのログを stderr、syslog、または両方に書き込むように設定できます。

アダプタのロギング

ロイター OMM アダプタは、Event Stream Processor と同じロギングのオプションをサポートします。

-d オプションは、デバッグ・レベルを設定します (0= 緊急メッセージのみ、7= すべてのメッセージ)。

-l オプションは、アダプタにログ・メッセージを stderr、syslog、または両方のいずれかに書き込むように指定します。いずれにも書き込まないようにも指定できます。 **-l** オプションを使用してアダプタにログ・メッセージを stderr に書き込むように指示する場合、stderr をファイルにリダイレクトすることをおすすめします。

入力アダプタ・マップ・ファイルの **publication** 要素の name 属性は、アダプタがどのように設定されているかを容易に識別できるようにするためにログ記録される説明的なテキスト文字列を指定します。たとえば、`subexample.xml` の行 3 〜 6 は、以下のように、入力アダプタ・マップ・ファイルのサブスクライブ元インスタンスの **publication** 要素を指定します。

```
<publication
    name="RMDS OMM Adapter"
    retryInterval="5"
/>
```

アダプタは Event Stream Processor に接続して対話するので、この設定を行うと、アダプタはログ・メッセージを以下のように書き込みます。

```
(0.123) @1 INFO: Configuring publication with name RMDS Adapter exp
```

最初の 2 つのフィールドは、それぞれタイムスタンプ (起動してからの秒数) とスレッド番号です。 タイムスタンプの基本時間と他の情報が、以下の例に示されているように、起動時にログ・ファイルに書き込まれます。 タイムスタンプを日付と時刻に変換するには、基本時間に秒数を単純に加算します。 これらのフィールドは、「ログ・メッセージ」トピックのサンプル・メッセージにはありません。

```
(63359098041.768) @1 NOTICE:Base time is 10/08/08-17:27:21
(0.001) @1 NOTICE:insta-a sub -c cimtest:-- -d 7
-f /home/sybase/support/1.0.3/ReutersOMMAdapter/marketprice.map.xml
-l 1 -p tigris:12192 -P 1
```

```
(0.001) @1 NOTICE:pid=28649
(0.001) @1 DEBUG:Using ESP_RMDSOMM_SUBSCRIBE_DEBUG_LEVEL=7ll/
i86pc_64_spro/bin/rmdsomm version:
1.0.3a-alpha_r18674M
```

ページ・データと部分的なページ更新

一部の Reuters データは、「部分的なページ」フォーマットを使用するページとして到着します。 各ページは、複数行で構成されており、最初はスナップショットとして送信されます。 ページ・データは、いずれの特別な設定も必要とせずにサポートされます。 アダプタ・ログ・ファイルからの以下の引用は、最初のページ・イメージ (強調表示されています) の配信を示しています。

```
(27.729) @6 INFO:Publishing VOD.mGBPd 21 of 21 on stream1 as UPSERT
_ITEM_NAME_ STRING: VOD.mGBPd
_SERVICE_NAME_ STRING: IDN_RDF
_SEQUENCE_NUMBER_ INT32: 1
_ITEM_STALE_ INT32: 0
ROW80_1 STRING: VOD.mGBPd SI Quote Publication
ROW80_2 STRING:
ROW80_3 STRING: DATE:03/07/2008 Time:11:09
ROW80_4 STRING:
ROW80_5 STRING: Time Venue SI Bid Size Bid Price Ask Price Ask Size
Status
ROW80_6 STRING: ==== ===== == ======== ========= ========= ========
======
ROW80_7 STRING: 110937 GSILGB2XXXX GSIL 1 150.9000 150.9500 1 OPEN
ROW80_8 STRING: 070021 SBILGB2LXXX CITI OPEN
ROW80_9 STRING: 110909 CSFBGB2LXXX CSFB 329 150.7000 151.1500 329
OPEN
ROW80_10 STRING: 110942 DEUTGB22ZEQ DBBL 528 150.6500 151.2000 527
OPEN
ROW80_11 STRING: 110946 ABNAGB22XXX ABNV 483306 150.9000 150.9500
483306 OPEN
ROW80_12 STRING: 110936 UBSWGB2LEQU UBSI 1 149.7682 152.1325 1 OPEN
ROW80_13 STRING: 110828 SBUKGB21XXX CITI 20600 150.9000 151.0000
20600 OPEN
ROW80_14 STRING: 110937 SLIIGB2LXXX LEHM 3750 150.9000 150.9500 15
OPEN
ROW80_15 STRING:
ROW80_16 STRING:
ROW80_17 STRING:
(27.730) @6 DEBUG:Immediate flush for low latency; opcode=p
```

ページの各行には、独自の FID があり、ページに対する行指向のデルタを促進します。 アダプタは、Reuters からの部分的なページ更新を解析し、アダプタ・ログ・ファイルからの以下の引用で強調表示されているような文字列を生成します。

```
(49.934) @6 DEBUG:Processing update for VOD.mGBPd from service
IDN_RDF
(49.934) @6 INFO:Publishing VOD.mGBPd 4 of 21 on stream1 as UPSERT
_ITEM_NAME_ STRING: VOD.mGBPd
_SEQUENCE_NUMBER_ INT32: 2
ROW80_3 STRING: off:78 size:2 value:10
```

```
ROW80_11 STRING: off:2 size:3 value:101
(49.934) @6 DEBUG:Immediate flush for low latency; opcode=p
(50.315) @6 DEBUG:Processing update for VOD.mGBPd from service
IDN_RDF
(50.315) @6 INFO:Publishing VOD.mGBPd 3 of 21 on stream1 as UPSERT
_ITEM_NAME_ STRING: VOD.mGBPd
_SEQUENCE_NUMBER_ INT32: 3
ROW80_11 STRING: off:5 size:1 value:7
(50.315) @6 DEBUG:Immediate flush for low latency; opcode=p
```

この例の最初の更新は、2 文字の文字列 "10" を、ROW80_3 FID からのデータで構成されるページの行の 78 文字目のオフセットに書き込みます。 例の 2 つ目の更新は、3 文字の文字列 "101" を、ROW80_11 FID からのデータで構成されるページの行の 2 文字目のオフセットに書き込みます。例の 3 つ目の更新は、1 文字の文字列 "7" を、ROW80_11 FID からのデータで構成されるページの行の 5 文字目のオフセットに書き込みます。 このように、ページに対する更新は、非常に簡潔です。

ログ・エントリ・フォーマットの変更
ログ・エントリのデフォルト・フォーマットを、2 つの方法で変更できます。

環境変数 ESP_RMDS_SUBSCRIBE_SYMBOL_FORMAT を 1 に設定して、Event Stream Processor に流れる値を示すメッセージをデフォルトの複数行フォーマットの代わりに単一行でログ記録するように、システムを設定できます。これによって、ログ・ファイルに書き込まれたメッセージに対して特定の項目をスキャンするのが容易になります。

double のデータ型の変数の出力で表示される小数点以下の桁数を指定できます。これを行うには、-P オプションを **esp_rmdsomm** コマンドで使用します。

デフォルトでは、Event Stream Processor に流れる値を示すログ・メッセージは、以下の例で示されているように、複数行フォーマットで書き込まれます。

```
(38079.526) @2 INFO:Publishing VOD.mGBPd 3 of 9 on stream1 as UPSERT
_ITEM_NAME_ STRING: VOD.mGBPd
_SEQUENCE_NUMBER_ INT32: 953
ROW80_7 STRING: off:53 size:2 value:45
```

環境変数 ESP_RMDS_SUBSCRIBE_SYMBOL_FORMAT を 1 に設定すると、これらのメッセージは、以下の例で示されているように、単一行フォーマットで書き込まれます。

```
(17.794) @5 DEBUG:stream1 p values: _ITEM_NAME_=VOD.mGBPd
_SEQUENCE_NUMBER_=2
ROW 80_3=off:78 size:2 value:20
```

-P オプションを使用して、以下の例の ask と last のように double のデータ型として宣言されている変数の表示方法を変更できます。 これは、変数の表示のみに影響し、内容には影響しません。

```
<RowDefinition id="omm_RowDef">
<Column name="symbol" datatype="string" />
```

```
<Column name="service" datatype="string" />
<Column name="seq" datatype="integer" />
<Column name="stale" datatype="integer" />
<Column name="bid" datatype="money" />
<Column name="ask" datatype="double" />
<Column name="last" datatype="double" />
<Column name="volume" datatype="integer" />
<Column name="when" datatype="timestamp" />
</RowDefinition>
```

デフォルトの精度を受け入れると、double のデータ型の変数 (たとえば、以下の例の ASK) は、小数点以下 3 桁で書き込まれます。

```
(5.089) @5 INFO:Publishing EURJPY= 7 of 9 on stream1 as UPSERT
(5.090) @5 DEBUG:stream1 p values: _ITEM_NAME_=EURJPY=
_SEQUENCE_NUMBER_=1 _ITEM_STALE_=0 BID=137.4800 ASK=137.530
ACVOL_1=0
ACTIV_DATE+TIMACT=2008-10-06T21:07:00.000 (1223327220000)
```

esp_rmdsomm コマンドの入力時にオプション -P 7 を指定すると、データ型 double の変数 (たとえば、以下の例の ASK) は、小数点以下 7 桁で書き込まれます。 他の型の変数は影響を受けません。

```
(4.913) @5 INFO:Publishing EURJPY= 7 of 9 on stream1 as UPSERT
(4.913) @5 DEBUG:stream1 p values: _ITEM_NAME_=EURJPY=
_SEQUENCE_NUMBER_=1 _ITEM_STALE_=0 BID=137.5200 ASK=137.5700000
ACVOL_1=0 ACTIV_DATE+TIMACT=2008-10-06T20:55:00.000 (1223326500000)
```

Reuters のロギング

Reuters 側の設定ファイルを使用して、Reuters のロギングを有効または無効にします。

ロイター OMM アダプタの RMDS へのインタフェースは、ロギング機能に書き込むようにも設定できます。 Reuters 側の設定ファイル (`rmdsomm.cfg` は、アダプタと共に提供される設定ファイル) で、ロギングを有効または無効にでき、また、ログ・ファイルのパスとファイル名を指定できます。 Reuters インタフェースも「メッセージ・ファイル」のセットをサポートします。

Reuters 側の設定ファイルには、Reuters の「ロガー」機能のための設定エントリのセットがあります。

```
¥Logger¥AppLogger¥fileLoggerEnabled                = true
¥Logger¥AppLogger¥fileLoggerFilename              = "rfasub.{p}.log"
```

この設定は、ロイター OMM アダプタ用の Reuters ロギングを有効にします。ログ・メッセージは、`rfasub.`**PID**`.log` ファイルに書き込まれます。ここで、**PID** はアダプタのプロセス ID です。

このセットの最初の行 ¥Logger¥AppLogger¥windowsLoggerEnabled = false は、Windows のロギング機能に対応しています。ただし、このロギング機能は、ロイター OMM アダプタに対してはサポートされません。
これらの例の行は rmdsomm.cfg からで、RMDS にサブスクライブするアダプタを設定します。

以下に示すように、同じファイルに、コンポーネント・ロガーのための設定エントリがあります。

```
¥Logger¥ComponentLoggers¥Connections¥messageFile   =¥
   "config/messages/RFA7_Connections.mc"
¥Logger¥ComponentLoggers¥Adapter¥messageFile       =¥
   "config/messages/RFA7_Adapter.mc"
¥Logger¥ComponentLoggers¥SessionCore¥messageFile   =¥
   "config/messages/RFA7_SessionLayer.mc"
¥Logger¥ComponentLoggers¥SSLED_Adapter¥messageFile =¥
   "config/messages/RFA7_SSLED_Adapter.mc"
```

ログ・メッセージ
これらの例は、アダプタ・ログ・ファイルからの典型的なエントリを示します。

ログ・メッセージの実際のフォーマットと動作は、ログ記録されるイベントの性質と、これらのイベントに関連付けられているログ・レベルも同じく、以降のアダプタのリリースで変更される可能性があります。

- **メッセージ：** – `NOTICE:Item BARC.VX is closed: No Quality of Service is available to process subscription, timeout expired`
- **原因：** – Reuters 設定ファイルの Reuters ユーザ名の値が不正です (大文字小文字の問題)。または、マップ・ファイルの Reuters サービス名が不正です。
- **メッセージ：** – `DEBUG: Immediate flush for low latency`
- **原因：** – RMDS から受信したデータがすぐに Event Stream Processor に送信されています。
- **メッセージ：** – `NOTICE:XMLRPC ERROR-116: The connection to the server could not be established. Please make sure the server is up, and check the specified host name/port, user name/password, and encryption settings. If a host name is specified, make sure that it can be resolved through a DNS lookup. (5.092) @1 INFO:Could not connect to SP; (tigris: 12190 cimtest) will retry in 5 seconds.`
- **原因：** – Event Stream Processor を実行しているサーバに接続できません。
- **メッセージ：** – `Ignoring market data event because no significant fields updated`

- **原因：** – アダプタは Reuters からデータを受信しましたが、いずれのフィールドも Event Stream Processor ストリームの対象になりませんでした。このため、データは何も送信されませんでした。
- **メッセージ：** `– ERROR: Error publishing: PUBLICATION ERROR-442: The send method of this publication object failed.`
- **原因：** – Event Stream Processor への接続がメッセージ転送時に失敗しました。
- **メッセージ：** `– ERROR:Mismatch between platform stream (9 columns) and adapter (31 columns for stream: stream1)`
- **原因：** – アダプタで定義されているカラムの数が、ストリームのカラム数と一致しませんでした。
- **メッセージ：** – `WARNING: Event Stream Processor down, dropping all subscriptions`

  次のようなメッセージが複数回繰り返して続きます。

  `DEBUG: Unsubscribing item: EUR= service: IDN_RDF`
- **原因：** – Event Stream Processor への接続が失われました。アダプタはどこにもデータを送信できないので RMDS データへのサブスクリプションを停止します。
- **メッセージ：** `–WARNING: Discarding data rec'd after unsubscribe`
- **原因：** – アダプタがサブスクリプションを遮断する前に、さらに多くのデータが到着しました。 このデータは無視されます。Event Stream Processor への接続がないためです。
- **メッセージ：** – `DEBUG: Processing update for EUR= from service IDN_RDF`
- **原因：** – サービス "IDN_RDF" 上の RIC "EUR=" 用の更新が到着しました。
- **メッセージ：** – `WARNING: Event Stream Processor down, dropping all subscriptions`

  次のようなメッセージが複数回繰り返して続きます。

  `DEBUG: Unsubscribing item: EUR= service: IDN_RDF`
- **原因：** – Event Stream Processor への接続が失われました。 アダプタはどこにもデータを送信できないので RMDS データへのサブスクリプションを停止します。
- **メッセージ：** `–WARNING: Discarding data rec'd after unsubscribe`
- **原因：** – アダプタがサブスクリプションを遮断する前に、さらに多くのデータが到着しました。 このデータは無視されます。Event Stream Processor への接続がないためです。

- **メッセージ：** – `EMERGENCY: Fatal Error at line 0, column 0 of config file: An exception occurred! Type:RuntimeException, Message:The primary document entity could not be opened. Id=/home/sybase/adapter/trunk/src/ReutersAdapter/xxsubexample.xml`
- **原因：** – 指定された設定ファイルが利用できません。
- **メッセージ：** – `EMERGENCY: Fatal Error at line 0, column 0 of config file: An exception occurred! Type:RuntimeException, Message:The primary document entity could not be opened. Id=/home/sybase/adapter/trunk/src/ReutersAdapter/xxsubexample.xml`
- **原因：** – 指定された設定ファイルが利用できません。

RTView アダプタ

Sybase Event Stream Processor RTView アダプタは、外部アダプタで、Event Stream Processor からのデータを RTView® Enterprise Dashboard に送信します。 このアダプタを使用するには、SL Corp. 提供の RTView Enterprise ソフトウェアが必要です。

このマニュアルでは、アダプタを使用するために RTView ソフトウェアを設定する方法についての情報を提供しますが、完全な詳細と最新の情報については、SL Corp. のマニュアルを参照してください。

RTView アダプタのデータ型のマッピング

Event Stream Processor のデータ型は、RTView のデータ型にマップされます。

Event Stream Processor のデータ型	SL RTView のデータ型
`boolean`	`boolean`
`integer`	`integer`
`long`	`long`
`float`	`double`
`date`	`date`
`timestamp`	`date`
`string`	`string`
`money, money(1)...money(15)`	`double`
`binary`	`string`

Event Stream Processor のデータ型	SL RTView のデータ型
`interval`	`long`
`bigdatetime`	`date`

RTView アダプタのインストール

RTView アダプタをインストールするには、RTView アダプタ・ファイルをアンパックし、アダプタと RTView ソフトウェア用の環境変数を設定します。

前提条件

- クライアント・マシンに、SL Corporation 提供の Enterprise RTView ソフトウェアのバージョン 5.8 または 5.9 のいずれかをインストールする。
- クライアント・マシンに、Java Software Development Kit 1.6 (またはそれ以降) をインストールする。
- JAVA_HOME 環境変数をインストール環境のルート・ディレクトリに設定する。
- ESP_HOME 環境変数を設定する。

手順

1. 環境変数 RTVIEWADAPTER_HOME を作成し、その値をフォルダ `$ESP_HOME¥adapters¥rtview` に設定します。
2. RTV_HOME 環境変数が Enterprise RTView インストール環境のロケーションに設定されていることを確認します。 この環境変数は、インストール時に自動的に設定されます。
3. RTView の `lib` フォルダと `bin` フォルダを PATH 環境変数に追加します。 たとえば、PATH を `$RTV_HOME/bin;$RTV_HOME/lib;$PATH` に更新します。

設定： Sybase 接続の作成と更新

サーバの接続情報を使用して、`ESPOPTIONS.ini` 設定ファイルを作成し、更新します。

サーバへの複数の接続を作成できます。 各接続は、特定のサーバ、ホスト、プロパティで構成されます。 サーバへの接続は、1 つ以上設定する必要があります。

1. Display Builder で、[Tools] > [Options] を選択します。
2. [Application Options] ウィンドウの左側のペインで、[ESP] を選択します。
3. [ESP Connections] タブで、[Add] をクリックします。

4. 既存の接続のプロパティを変更するには、接続をダブルクリックします。
5. 該当する接続情報をすべて入力して、[OK] をクリックします。
6. [Apply] をクリックし、次に [Save] をクリックして、接続情報を `ESPOPTIONS.ini` 設定ファイルに保存します。 接続ファイルをアダプタ・インストール環境の `lib` フォルダに保存するかどうかを確認するメッセージが表示されたら、[No] をクリックします。 これによって、接続情報が現在のプロジェクトのみに適用されます。

次のステップ

接続を作成し、編集したら、RTView Display Builder またはサーバのいずれかを再起動して、変更を反映します。

Event Stream Processor のパラメータ

Event Stream Processor への接続を作成するために `ESPOPTIONS.ini` 設定ファイルで指定できるパラメータ。

パラメータ	タイプ	説明
authType	`choice`	(必須) Event Stream Processor への認証に使用される方法を指定。 デフォルト値はありません。 • UserPassword – **user** パラメータと **password** パラメータが必須。 • ServerRSA – **user** パラメータ、**keyStore** パラメータ、**keyStoreFile** パラメータが必須。 • None。
projectUri	`string`	(オプション) Event Stream Processor クラスタに接続するための完全なプロジェクト URI を指定。たとえば、`esp://hostname:port/workspace/project` のように記述します。 デフォルト値はありません。
keyStore	`string`	(オプション) RSA キーストアのロケーションを指定し、パスワード値を復号。 **authType** が ServerRSA に設定されている場合は必須です。 デフォルト値はありません。
keyStorePassword	`string`	(オプション) キーストアのパスワードを指定し、パスワード値を復号。 **authType** が ServerRSA に設定されている場合は必須です。 デフォルト値はありません。

パラメータ	タイプ	説明
user	`string`	(オプション) Event Stream Processor にログインするために必要なユーザ名を指定。 None 以外の認証スキームで必須です (**authType** を参照してください)。 デフォルト値はありません。
password	`string`	(オプション) Event Stream Processor にログインするために必要なパスワードを指定。 UserPassword 認証スキームで必須です (**authType** を参照してください)。 デフォルト値はありません。
isEncrypted	`string`	(オプション) パスワードを暗号化するかどうかを指定。 有効な値は、true または false です。 true に設定すると、password は暗号化されたフィールドになります。 これによって、サーバは暗号化されたテキストとしてパスワードを認識し、実行時にパスワードを復号できるようになります。デフォルト値は true です。
getBase	`string`	(オプション) サブスクリプションのセットアップ時にサーバがストリームに既存データを送信するかどうかを指定。 有効な値は、true または false です。 true に設定すると、サーバは、ストリームに既存データを送信しません。デフォルト値は false です。
droppable	`string`	(オプション) そのクライアントが起動状態を維持できない場合にサーバがこの接続を破棄するかどうかを指定。 true に設定すると、サーバは接続を破棄します。デフォルト値は false です。
lossy	`string`	(オプション) そのクライアントが起動状態を維持できない場合にサーバがレコードを破棄するかどうかを指定。 true に設定すると、サーバはレコードを破棄します。デフォルト値は false です。
shineThrough	`string`	(オプション) Event Stream Processor が、変更されていないフィールドのデータを送信するかどうかを指定。 true に設定すると、サーバはデータを送信しません。デフォルト値は false です。

パラメータ	タイプ	説明
refreshInterval	integer	(オプション) パルス間隔を指定。 Event Stream Processor は、データを統合し、このデータを、ミリ秒単位で指定された間隔で定期的に送信します。 0 以下の値を指定すると、パルス出力が無効になります。 デフォルト値はありません。
dateFormat	string	(オプション) 日付フォーマットを指定。 デフォルト値は、YYYY-MM-DDHH24:MI:SS:FF です。
timestampFormat	string	(オプション) タイムスタンプ・フォーマットを指定。 デフォルト値は、YYYY-MM-DDHH24:MI:SS:FF です。
delimiter	string	(必須) パブリッシャのコマンド・ラインで使用。 デフォルト値は、## です。
defaultconnection	string	(必須) サーバへの接続に使用する接続文字列を指定。 デフォルト値は、conn1 です。
retryInterval	integer	(必須) 接続が切断された場合に、サーバに再接続するために待機する時間を指定。デフォルト値は 0 です。
pollinterval (秒)	integer	(必須) データのポーリングを待機する期間を指定。デフォルト値は 0 です。

オペレーション

RTView アダプタをインストールしたら、Display Builder と Display Viewer を使用できるようになります。

RTView Display Builder の起動

ダッシュボード・プロジェクトを構築して実行するには、コマンド・ラインから Display Builder を起動します。

前提条件

サーバを起動します。

手順

コマンド・ラインで、アダプタ・インストール・フォルダから Display Builder を実行すると、Builder は起動時にサーバとのリンクを確立します。

各ダッシュボード・プロジェクトを独自のフォルダに格納することをおすすめします。 以降で、このフォルダから Display Builder を起動できます。

1. 新しいダッシュボード・プロジェクトを起動するには、そのための新しいフォルダを作成します。 既存のプロジェクトを開くには、[スタート]>[ファイル名を指定して実行] を選択します。
2. 次を入力して、RTView Display Builder でプロジェクトを起動します。

```
%RTVIEWADAPTER_HOME%¥bin¥start_builder.bat <project_filepath>
[<rtv_file_name>.rtv]
```

   - `<project_filepath>` はプロジェクト・フォルダの絶対ファイル・パス。
   - `<rtv_file_name>` は、既存のダッシュボードの名前。 新しい .rtv ファイルを作成するときには、ファイル名を指定しないでください。 Display Builder は、`unnamed.rtv` と呼ばれるブランクのダッシュボードを開きます。この後に、手順 1 で作成した新しいダッシュボード・プロジェクトのフォルダに希望する名前を使用して保存できます。

注意： Windows では、start builder コマンドを入力し、ファイル名なしでプロジェクト・フォルダのみを使用すると、"<projectfolder>.rtv" という名前のファイルが検索されます。 そのようなファイルがフォルダに存在しない場合、ファイルが開けないことを示すメッセージが表示されます。 [OK] をクリックして、Builder を起動します。 これは、RTView の既知の問題です。

RTView Display Viewer の起動

実行時データの表示を開始するには、Display Viewer をコマンド・ラインから起動します。

前提条件

サーバを起動します。

手順

コマンド・ラインでアダプタのインストール・フォルダから Display Viewer を実行すると、Viewer が起動時にサーバにリンクされます。

各ダッシュボード・プロジェクトを独自のフォルダに格納することをおすすめします。 その後、このフォルダから Display Viewer を起動できます。

1. 新しいダッシュボード・プロジェクトを起動するには、そのための新しいフォルダを作成します。 既存のプロジェクトを開くには、[スタート]>[ファイル名を指定して実行] を選択します。
2. 次のように入力して、RTView Display Viewer でプロジェクトを起動します。

```
%RTVIEWADAPTER_HOME%¥bin¥start_viewer.bat <project_filepath>
<rtv_file_name>.rtv
```

- `<project_filepath>` は、プロジェクト・フォルダの絶対ファイル・パス。
- `<rtv_file_name>` は、ダッシュボードの .rtv ファイルの名前。 Viewer を起動するには、これを指定する必要があります。

ダッシュボード・プロジェクトのショートカットの作成

適切なロケーションから Display Builder または Display Viewer を起動します。

ショートカットは、Builder または Viewer を起動し、同時に、指定されたダッシュボード・プロジェクトを開きます。

1. ショートカットを作成するロケーションで、メニュー・バーから [File] > [New] > [Shortcut] を選択します。または、右クリックして、[New] > [Shortcut] を選択します。
2. Create Shortcut ウィザードで、ショートカットに名前を割り当ててから、項目のロケーションとして `%RTVIEWADAPTER_HOME%¥bin ¥start_builder.bat <project_filepath> [<rtv_file_name>.rtv]` を入力します。

   `<project_filepath>` は、プロジェクト・フォルダの絶対ファイル・パスです。`<rtv_file_name>` は、開くダッシュボードの .rtv ファイルの名前です。

   [次へ] をクリックします。
3. ショートカットを右クリックし、[Properties] を選択します。
4. [Properties] ウィンドウで、[Run] フィールドを [Minimized] に設定します。
5. 手順 2 ～ 4 を繰り返して、Display Viewer でダッシュボード・プロジェクトを起動する、対応するショートカットを作成します。

   項目のロケーションとして `%RTVIEWADAPTER_HOME%¥bin ¥start_viewer.bat <project_filepath> <rtv_file_name>.rtv` を入力します。

ダッシュボード・オブジェクトとデータ・ストリーム

ほとんどの RTView ダッシュボード・オブジェクトを Event Stream Processor のデータ・ストリームに接続できます。 この接続を使用することによって、ダッシュボード・オブジェクトは、ストリームからリアルタイム・データを受信できます。

ストリームに接続するには、ストリームのタイプに応じて、2 つの方法が利用できます。

- ストリームがキー付きエントリに対して更新と削除を生成する場合、最初に、キャッシュと呼ばれる中間オブジェクトをセットアップする必要があります。その後に、ダッシュボード・オブジェクトを接続できます。

- ストリームに時系列などの挿入要素がある場合、ダッシュボード・オブジェクトをストリームに直接接続します。

キャッシュの作成

RTView Builder を使用して別の .rtv ファイルにキャッシュを作成し、そのファイルをメイン・ダッシュボード・ファイルにインポートします。

キャッシュはデータソースの 1 つで、リアルタイム・データの高速な分析処理と、履歴データに対する現在のリアルタイム値の比較を可能にします。 キャッシュは、ダッシュボード・オブジェクトを、ユーザ入力値に対する更新と削除を生成する Event Stream Processor のデータ・ストリームに接続するときの中間データソースです。 RTView Builder から、次を実行します。

1. [File] > [New] を選択して、新しい .rtv ファイルを作成します。
2. [Tools] > [Caches] を選択し、次に [Display Builder] メイン・ウィンドウの下部の [Caches] タブで [Add] をクリックします。
3. 新しいキャッシュの名前を入力し、タイプを [Table] に設定します。
4. キャッシュのプロパティを編集します。

プロパティ	プロシージャ
valueTable	1. [valueTable] を右クリックし、[Attach To Data] > [ESP] を選択。 2. キャッシュがサブスクライブするストリームの名前を選択。 3. (オプション) 特定のストリーム・カラムを選択。ただし、特定のカラムを選択する場合、プライマリ・キー・カラムを選択する必要があります。
indexColumnNames	1. [indexColumnNames] フィールドの隣の省略記号 ([...]) をクリック。 2. ストリームのキー・カラムを指定。複数のカラム名を指定する場合、セミコロンで区切ります。 **注意：** カラム名は、大文字と小文字が区別されます。
deleteTable	deleteTable はデータ・アタッチメントで、指定されたローをキャッシュ・テーブルから除去。 ローは、インデックス・カラム値が、このアタッチメントから受信されるテーブル・データ内のローの値と一致する場合に除去されます。 1. [deleteTable] を右クリックし、[Attach To Data] > [ESP] を選択。 2. Event Stream Processor には、[deleteTable] がないので、仮想テーブル名を使用する必要あり。 たとえば、Positions ストリーム名を使用するには、deleteTable プロパティに !Positions を使用します。

プロパティ	プロシージャ
maxNumberOfRows	• 保存する履歴データのロー数を指定。 デフォルト値はゼロです。 ゼロより大きい数を指定すると、履歴データの格納が有効になります。 すべてのキー値に対して、履歴データの N ローが維持されます。ここで、N は指定されたロー数です。

5. ファイルを保存します。
6. このファイルをメイン・ダッシュボード・ファイルにインポートします。 メイン・ダッシュボード・ファイルから次を実行します。
   a) [Tools] > [Options] > [Caches] を選択します。
   b) [Add] をクリックし、手順 3 で作成したキャッシュを選択します。
   c) [適用] をクリックし、次に [OK] をクリックします。

   注意： キャッシュをメイン・ダッシュボード・ファイルにインポートした後に変更を加えるには、[Options] > [Caches] > [Refresh Selection] を選択します。
7. ファイルを保存して閉じます。
8. 作成するキャッシュごとに、手順 1 ～ 7 を繰り返します。

例： オブジェクトをキャッシュに付加
ダッシュボード・テーブル・オブジェクトを以前に作成したキャッシュに付加します。

キー付きエントリに対して更新または削除を生成するストリームに、ダッシュボード・オブジェクトを直接接続することはできません。 キャッシュをストリームに接続し、その後に、オブジェクトをキャッシュに接続します。

オブジェクトをストリームに、直接またはキャッシュを介して付加します。これを行うには、[Object Properties] ペインの [Data] 見出しの下の、オブジェクトの値プロパティを設定します。 テーブル・オブジェクトの場合、このプロパティは valueTable になります。

この例では、最初に、テーブルをダッシュボードに付加し、次に、valueTable プロパティを使用して、以前に作成され、Event Stream Processor ストリームに接続するキャッシュに付加します。

1. [Object Palette] の [Tables] タブからテーブルを選択し、キャンバスをクリックしてテーブル・オブジェクトをダッシュボードに追加します。
2. キャッシュをダッシュボード・プロジェクトにインポートします。
   a) [Tools] > [Options] > [Caches] を選択します。
   b) [Add] をクリックし、キャッシュを選択します。
   c) [適用] をクリックし、次に [OK] をクリックします。

3. テーブル・オブジェクトの valueTable プロパティを右クリックし、[Attach To Data] > [Cache] に移動します。
   これによって、データソースを設定するためのダイアログ・ボックスが開きます。
4. 手順 2 からのキャッシュを選択します。
5. [Table] フィールドの [Current] を選択し、表示するカラムを選択します。
6. (オプション) [Filter Rows to Basic or Advanced] を選択して、データのサブセットを表示します。
7. [適用] をクリックし、次に [OK] をクリックします。

例： オブジェクトをストリームに付加
ダッシュボード・テーブル・オブジェクトを、挿入のみがあるストリームに直接付加します。

オブジェクトをストリームに、直接またはキャッシュを介して付加します。これを行うには、[Object Properties] パネルの [Data] 見出しの下の、オブジェクトの値プロパティを設定します。 テーブル・オブジェクトの場合、このプロパティは valueTable になります。

1. [Object Palette] の [Tables] タブからテーブルを選択し、キャンバスをクリックしてテーブル・オブジェクトをダッシュボードに追加します。
2. テーブル・オブジェクトの valueTable プロパティを右クリックし、[Attach To Data] > [ESP] を選択します。
3. 使用する接続、表示するテーブルとカラムを選択します。
4. 必要に応じて、ストリームのローとカラムをフィルタしてデータのサブセットを表示します。

例： 関数の作成
リスト・ボックス・オブジェクト (obj_c1tlb) を設定するテーブル値のリストを返す関数を作成します。

関数を使用すると、テーブル・カラムの平均値の算出や複数データ・アタッチメントの値の追加など、共通の計算を自動化できます。

それぞれスカラ・データと表データを対象にするスカラ関数と表関数の 2 つのカテゴリーにグループ化される関数がいくつかあります。

Display Builder から、次を実行します。

1. [Tools] > [Functions] を選択し、[Display Builder] ウィンドウの下部にある [Functions] タブをクリックして、[Add] をクリックします。
2. 関数の名前を入力し、該当する関数タイプを選択します。 たとえば、テーブルまたはストリームのカラムから一意な値のリストを返すために、[Count

Unique Values] を選択します。 この関数を選択すると、テーブルまたはストリームのカラムを問い合わせるメッセージが表示されます。

3. [Table Column] フィールドを右クリックし、[Attach To Data] > [ESP] を選択します。 Event Stream Processor ストリーム名への接続とストリームで定義されているカラムを選択します。
4. [Object Palette] の [Controls] タブで、リスト・ボックス・オブジェクトを作成します。
5. リスト・ボックス・オブジェクト・プロパティの [listValues] プロパティを右クリックし、[Attach To Data] > [Functions] を選択します。 手順 2 で作成した関数を選択し、[Columns] フィールドの [Value] を選択して、関数をリスト・ボックス・オブジェクトにバインドします。
   関数が付加されたストリームのカラムに新しい値が出現すると常に、その値は、リスト・ボックス・オブジェクトにも自動的に出現します。
6. リスト・ボックス・オブジェクトの `selectedValue` プロパティと `varToSet` プロパティを設定し、リスト・ボックスで行われた選択が、Event Stream Processor へのパブリッシュなど他の場所でも使用できるようにします。

Event Stream Processor へのパブリッシュ

ユーザ・コントロール・オブジェクトをダッシュボード・プロジェクトに追加して、Event Stream Processor ストリームをアクティブに操作できるようにします。

RTView アダプタは、ダッシュボードから Event Stream Processor へのデータのパブリッシュをサポートします。 ダッシュボードは、サーバの入力ストリームに RTView アダプタを介してイベントを送信します。

1. 1 つ以上のコントロール・オブジェクトをダッシュボードに追加します。
   設定するフィールドごとにコントロールを追加します。 入力ボックス、候補リスト、リスト・ボックス、チェック・ボックス、ボタンを使用できます。
   a) Display Builder の [Tools] メニューを使用して、各フィールドの変数を作成します。
   b) 各コントロールを変数に付加します。
   c) [Tools] > [Variables] を選択して変数を追加し、変数の名前、初期値、型を指定します。
   d) (オプション) [Tools] > [Functions] を選択して、コントロールでのユーザ選択を判断する関数を作成し、コントロールをその関数に付加します。
   e) (オプション) ダッシュボード上の他のオブジェクトのプロパティを、コントロールにリンクします。
2. アクション・コマンドをコントロール・オブジェクトに関連付けます。これを行うには、[Interaction] カテゴリーの下の [actionCommand] を右クリックし、[Define Command] > [ESP] を選択します。 パブリッシュ・コマンドを入力します。

```
conn_name.publish ## opcode ## stream_name ## col_value_1 ##
col_value_2 ...
```

パラメータ	説明
##	引数の区切り文字。任意の文字に設定できます。 デフォルト値は、## です。 [Tools] > [Options] を選択し、左側のペインで [ESP] を選択して、[Add] ボタンをクリックします。 区切り文字の前後には、スペースを挿入する必要があります。
conn_name	パブリッシュ用に使用される、事前に定義された接続。
opcode	実行するオペレーション。 有効な値は、以下のとおりです。 • 1 — INSERT • 3 — UPDATE • 5 — DELETE • 7 — UPSERT • 13 — SAFE DELETE
stream_name	ターゲット・ストリームの名前。
col_value_...	カラムの値。 カラム値の数は、ターゲット・ストリームのカラム数に一致する必要があります。 何も囲まずに 2 つの一重引用符を入力することによって、null 値を指定できます (たとえば、'')。 デフォルト値は、'' です。 RTView Viewer は、空のフィールドを 2 つの一重引用符 ('') としてパブリッシュします。 デフォルトの nullValue プロパティも '' なので、NULL が Event Stream Processor の対応するカラムに挿入されます。 空の文字列を挿入するには、[Add ESP Connection] ウィンドウで Null Value プロパティを別の値に変更します。 文字列以外のデータ型の場合、何も囲まない 2 つの一重引用符 (たとえば、'') は、常に null 値を表します。

- [Add ESP Connection] ウィンドウの Date format プロパティと Timestamp format プロパティと同じフォーマットで、日付値とタイムスタンプ値を指定。 これは、Java SimpleDateFormat オブジェクトが使用するのと同じフォーマット仕様です。
- パブリッシュ・コマンドが成功すると、応答はない。 コマンドが失敗した場合は、該当するエラー・メッセージが表示されます。

パブリッシャ例の実行

RTView Display Viewer を使用して、RTView アダプタと共に提供されるパブリッシャ例を実行します。

1. %RTVIEWADAPTER_HOME%¥examples にある、提供されている rtviewadapter.ccx モデルとポート 19011 を使用して Event Stream Processor を起動します。

オペレーティング・システム	手順
Windows	1. cd %RTVIEWADAPTER_HOME%¥examples を入力。 2. サンプル・クラスタを起動。 start_server_cluster.bat 3. プロジェクトをクラスタに追加し、クラスタ上で起動。 start_project.bat
UNIX	1. cd $RTVIEWADAPTER_HOME/examples を入力。 2. サンプル・クラスタを起動。 start_server_cluster.sh 3. プロジェクトをクラスタ上で起動。 start_project.sh

2. RTView Display Viewer をコマンド・ラインから起動します。

オペレーティング・システム	手順
Windows	%RTVIEWADAPTER_HOME%¥bin ¥start_viewer.bat %RTVIEWADAPTER_HOME%¥examples publisher.rtv
UNIX	$RTVIEWADAPTER_HOME/bin/ start_viewer.sh $RTVIEWADAPTER_HOME/ examples publisher.rtv

注意： **start_viewer** コマンドを **start_builder** コマンドに置き換えると、RTView Display Builder を起動できます。

3. 画面に表示される指示に従います。
4. データがサーバに正常にパブリッシュされたことを確認します。

オペレーティング・システム	手順
Windows	cd %RTVIEWADAPTER_HOME%¥examples run_sub.bat

オペレーティング・システム	手順
UNIX	`cd $RTVIEWADAPTER_HOME/examples` `run_sub.sh`

サブスクライバ・サンプル・コードの実行

RTView アダプタに付属のサブスクライバ・サンプル・コードを実行するには、RTView Display Viewer を使用します。

1. `%RTVIEWADAPTER_HOME%¥examples` に提供されている `rtviewadapter.ccx` モデルと共に Event Stream Processor を起動し、ポート 19011 を使用します。

オペレーティング・システム	手順
Windows	1. `cd %RTVIEWADAPTER_HOME%¥examples` と入力 2. サンプル・クラスタを起動。`start_server_cluster.bat` 3. プロジェクトをクラスタに追加し、クラスタ上で起動。`start_project.bat`
UNIX	1. `cd $RTVIEWADAPTER_HOME/examples` と入力。 2. サンプル・クラスタを起動。`start_server_cluster.sh` 3. プロジェクトをクラスタ上で起動。`start_project.sh`

2. RTView Display Viewer をコマンド・ラインから起動します。

オペレーティング・システム	手順

Windows	`%RTVIEWADAPTER_HOME%¥bin`
	`¥start_viewer.bat`
	`%RTVIEWADAPTER_HOME%¥examples`
	`subscriber.rtv`

UNIX	`$RTVIEWADAPTER_HOME/bin/`
	`start_viewer.sh $RTVIEWADAPTER_HOME/`
	`examples subscriber.rtv`

注意： **start_viewer** コマンドを **start_builder** コマンドに置き換えると、RTView Display Builder を起動できます。

3. **esp_convert** ユーティリティと **esp_upload** ユーティリティを使用し、`input.xml` データをロードし、XML データをストリーム・データに変換して、このデータをターゲット・ストリームに送ります。

オペレーティング・システム	手順
Windows	`cd %RTVIEWADAPTER_HOME%¥examples` `run_loaddata.bat`
UNIX	`cd $RTVIEWADAPTER_HOME/examples` `run_loaddata.sh`

4. (オプション) データが正常にストリームにロードされていることを確認します。

オペレーティング・システム	手順
Windows	`cd %RTVIEWADAPTER_HOME%¥examples` `run_sub.bat`
UNIX	`cd $RTVIEWADAPTER_HOME/examples` `run_sub.sh`

5. テーブルに表示されるストリーム・データを確認します。

既知の制限事項：

RTView アダプタの既知の制限事項を以下に示します。

- サーバに既に接続されている接続を変更するには、Builder またはサーバのいずれかを再起動する必要がある。
- ピリオドのある値はパブリッシュできない。 倍精度と通貨型の回避策として、ピリオドの代わりにカンマを小数点として入力できます。 `string` データ型については、解決方法がありません。
- RTView の `double` データ型は Event Stream Processor の `money` データ型にマップされるので、15 桁を超えるデータでは精度が失われる可能性がある。
- RTView アダプタは、`date` データ型のマイクロ秒精度を処理しない。 このため、RTView の `date` データ型は、Event Stream Processor の `bigdatetime` データ型にミリ秒の精度でマップされます。 これは、SL Corporation によって解決される必要のある RTView 制限です。

- [Add ESP Connection] ウィンドウで接続名を入力せずに [OK] をクリックすると、空のボックスが表示される。

Tibco Rendezvous アダプタ

アダプタのタイプ： tibcorvplugin。 Sybase Event Stream Processor の TIBCO Rendezvous アダプタは、サブスクライブし、Event Stream Processor から Rendezvous サーバにデータをパブリッシュします。

Rendezvous アダプタは、以下の機能を提供します。

- Rendezvous サーバに接続し、セッションを開いて、サブジェクトへのサブスクライブとサブスクリプションの削除を実行する。
- Rendezvous メッセージと Event Stream Processor レコードとの間で、相互に変換する。
- Rendezvous サーバとの間で、メッセージの受信とパブリッシュを実行する。

参照：

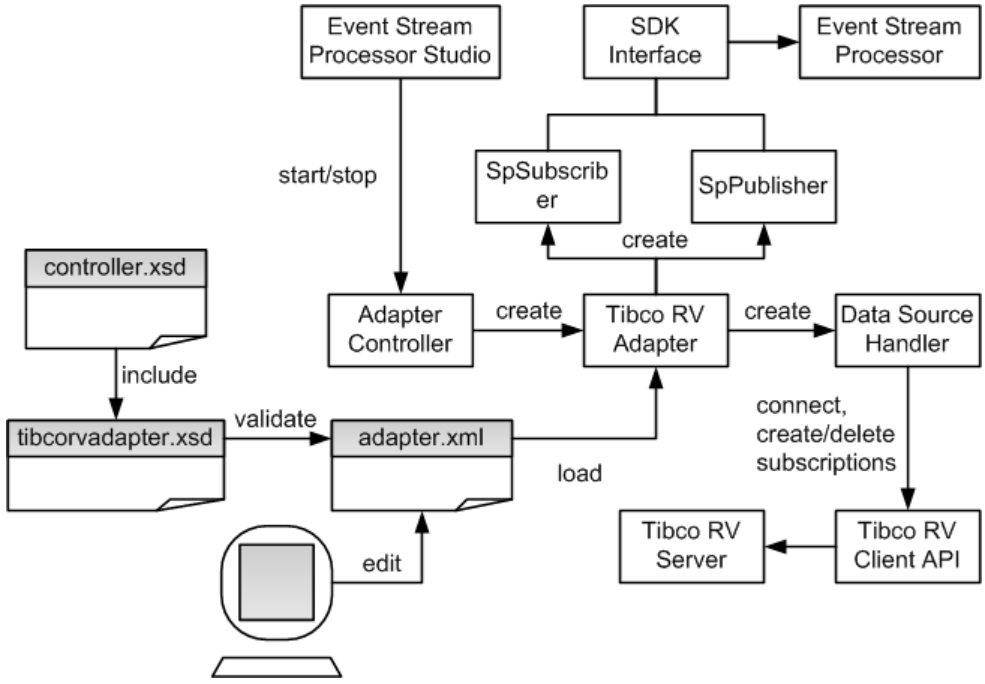
- *第 5 章、「保証された配信」* (591 ページ)

制御フロー

アダプタは設定をファイル (たとえば、`adapter.xml`) からロードし、アダプタ・スキーマ (`tibcorvadapter.xsd`) に対して検証します。このスキーマは、API 全体のコントローラ・スキーマ (`controller.xsd`) で構成されます。

スキーマは編集できません。

図 11 : TIBCO アダプタ制御フロー



アダプタ・コントローラは、アダプタのインスタンスを作成し、その後、ユーザ・コマンドを受け取り、実行します。 アダプタ・コントローラは、**start**、**stop**、**status** のコマンドを実行できます。

start コマンド

start コマンドは、アダプタの「コマンドと制御」のインタフェースを設定し、開始します。

Data Source Handler が、Rendezvous Client API を使用して、Rendezvous サーバに接続してセッションを開始します。 SpSubscriber と SpPublisher のコンポーネントが、Pub/Sub インタフェースを介して Event Stream Processor に接続します。 SpSubscriber はアウトバウンド・ストリームの受信を開始し、SpPublisher はデータをインバウンド・ストリームにパブリッシュする準備を整えます。

アダプタの実行中のインスタンスが存在するときに **start** コマンドを実行すると、コマンドは無視され、警告が送信されます。

参照：

- *TIBCO Rendezvous アダプタの起動* (543 ページ)

stop コマンド

stop コマンドは、SpPublisher と SpSubscriber を Event Stream Processor から切断します。これによって、Data Source Handler はセッションを閉じ、データソースから切断します。さらに、アダプタ・コントローラがユーザ・コマンドの受信を停止し、アダプタ・プロセスを終了します。

アダプタの実行中のインスタンスが存在しないときに **stop** コマンドを実行すると、コマンドは無視され、警告が送信されます。

参照：

- *TIBCO Rendezvous アダプタの停止* (545 ページ)

status コマンド

status コマンドは、アダプタのステータスを報告します。アダプタ・コントローラが次のステータスを表示します。 実行中または停止中のいずれか。

参照：

- *TIBCO Rendezvous アダプタのステータスの確認* (544 ページ)

データ・ストリーム

アダプタは、各 Rendezvous メッセージをストリーム・レコードに格納します。

単一のストリームが、異なるサブジェクトにメッセージを格納することがあります。 サブジェクトは、必須のカラム Subject に格納されます。 その他のカラムは、Rendezvous メッセージのフィールドに対応します。

ストリーム・カラムの名前が、Rendezvous メッセージの対応するスカラ・フィールドの名前に一致していることを確認します。 メッセージ型フィールドの場合、カラムは次の命名規則に従います。

<メッセージ型フィールド名>_<フィールド名>

アダプタは、任意の深度の埋め込みメッセージをサポートします。 Rendezvous メッセージに関連しないカラムは、許可されません。配列型のフィールドもサポートされません。

Client カラムと Date カラムは、スカラ・フィールドに対応します。 Trade は、Price と Volume の 2 つのフィールドで構成される埋め込みメッセージです。

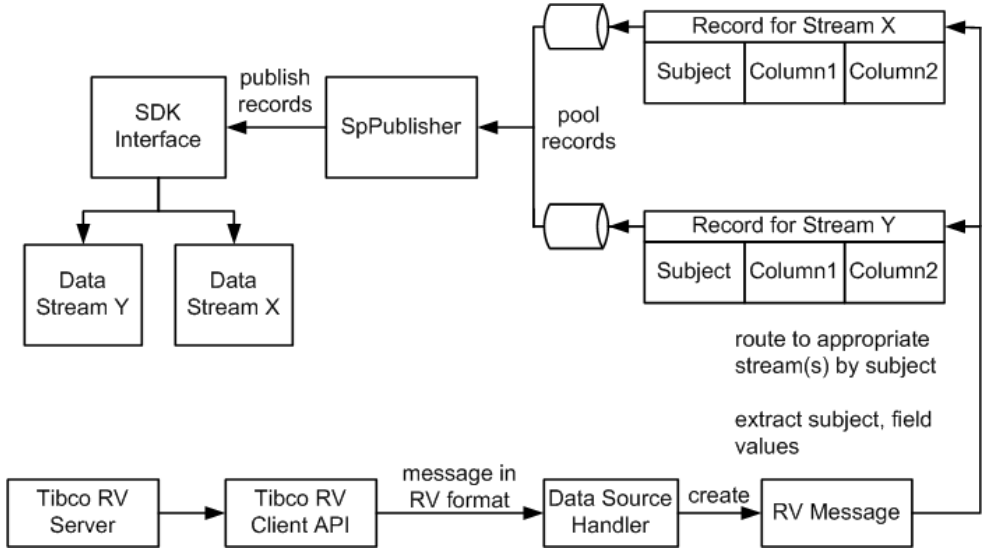
表 12 : サンプル・データ・ストリーム

サブジェクト	クライアント	日付	Trade_Price	Trade_Volume
MySubject	UBS	2008-03-13T08:19:30	34.7	6000

メッセージ・フロー

start コマンドは、アダプタを介するメッセージ・フローを開始します。

次の図は、インバウンドのメッセージ・フローを示しています。



アダプタは起動時に、設定ファイルの inbound セクションにリストされているすべてのサブジェクトにサブスクライブします。 ワイルドカードを使用したサブスクリプションがサポートされています。 インバウンド・メッセージは、クライアント API コールバックを介して受信されます。 次に、メッセージは、Message Distributor に渡されます。

Message Distributor は、各 Rendezvous メッセージを 1 つ以上のデータ・ストリーム向けのレコードに変換します。 レコードは、この時点で Event Stream Processor にパブリッシュするための準備が整いますが、すぐにはパブリッシュされません。 レコードはキューに登録され、個別のスレッド上の SpPublisher オブジェクトによって取り出されます。レコード・キューごとに 1 つのスレッドがあります。 キュー容量は設定できます。 キューのサイズを大きくすると、突発的に大量のメッセージが発生してもオーバフローが起こりにくくなります。 キューに登録されているレコードがキューの容量の 4 分の 3 を超えると、警告がログ記録されます。 容量が 4 分の 3 未満に戻ると、別の警告がログ記録されます。 キューがいっ

ぱいになると、次のレコードを登録できるようになるまで、アダプタは待機します。

レコードは非同期にパブリッシュされます。 アダプタは、いかなるフィードバックも Event Stream Processor から受信しません。 フェールオーバが発生すると、Pub/Sub API はメッセージを失うことなく、設定に従って、パブリッシュ先を予備の Event Stream Processor インスタンスに切り替えます。

注意： アウトバウンド・レコードの場合、opcode (ESP_OPS の値) は、TIBCO Rendezvous サーバに送信されません。 インバウンド・レコードの場合、すべてのレコードは opcode として "p" (upsert) が設定されてから、サーバにパブリッシュされます。

TIBCO Rendezvous アダプタのデータ型のマッピング

Event Stream Processor のデータ型は、TIBCO Rendezous のデータ型にマップされます。

Event Stream Processor のデータ型	TIBCO のデータ型
`boolean`	TibcorvMsg.Bool
`integer`	TibcorvMsg.i32
long	TibcorvMsg.i64
float/money/money1-money15	TibcorvMsg.f64
string	TibcorvMsg.string
date/timestamp	TibcorvMsg.datetime
bigdatetime	TibcorvMsg.i64
interval	TibcorvMsg.i64
binary	TibcorvMsg.string

JAVA_HOME 環境変数の設定

Java ディレクトリを指すように JAVA_HOME 環境変数を設定します。

前提条件

- Java Runtime Environment のバージョン 1.6.0_26 以降をインストールする。
- アダプタを実行しているオペレーティング・システム用の TIBCO Rendezvous バイナリ・ライブラリをインストールする。 バイナリ・ライブラリは、TIBCO Rendezvous アダプタ製品には同梱されません。

手順
JAVA_HOME 環境変数を、Java Runtime Environment 1.6.0_26 以降がインストールされているディレクトリ・パスに設定します。

次のステップ
- TIBCO Rendezvous バイナリ・ライブラリを `$ESP_HOME/adapters/tibco_rv/lib/tibco/<platform_type>` に配置する。ここで、`<platform>` は UNIX ベース・システムの **arch** コマンドによって取得されます。 Windows の場合、win32 と win64 のフォルダがあります。 オペレーティング・システムに応じたフォルダを使用して、ライブラリをコピーします。
- ESP_HOME 環境変数が正しく設定されていることを確認する。

設定
TIBCO Rendezvous アダプタ用の設定情報を以下に示します。

TIBCO Rendezvous アダプタのディレクトリ
アダプタのディレクトリには、設定ファイル、テンプレート、例、JAR ファイルなど、アダプタに関連するすべてのファイルがあります。

```
README.txt  Quick Guide
ReleaseNotes.txt  Release Notes

bin/
    adapter.bat  Standalone adapter startup script
    adapter.sh      Standalone adapter startup script
    adapter-plugin.bat Plug-in connector startup script
    adapter-plugin.sh  Plug-in connector startup script

config/
    controller.xsd     Controller schema
    log4j.properties   Sample logging configuration
    tibcorvadapter.xsd  Adapter schema
    login.config  Authentication configuration

example/        Working example
    GD       Working example for guaranteed delivery

lib/
    tibco/
        i86pc
        sun4
        win32
        win64
        x84_64
    esp_tibco_rv_adapter.jar  Tibco Rendezvous adapter library
    tibrvj.jar        Tibco Rendezvous java library

javadoc/
    adapterapi/     Adapter API Javadoc
```

```
    tibcorvadapter/     TIBCO Rendezvous Adapter Javadoc

Common jars are located:

$ESP_HOME/adapters/jar

activation.jar                Java mail library
adapterapi.jar                Adapter API code
axis.jar                     Webservices jar
commons-codec-1.3.jar         Required by SDK API
commons-discovery-0.2.jar
commons-logging-1.1.jar    Logging library
esp_java_sdk-0.4.jar     ESP SDK library
jaxrpc.jar                Required by ESP SDK
log4j-1.2.14.jar          Logging library
mail.jar              Java mail library
saaj.jar              Webservices jar
ws-commons-util-1.0.2.jar  Required by ESP SDK
wsdl4j-1.5.1.jar
xercesImpl.jar         XML parser library
xmlrpc-client-3.1.3.jar Required by ESP SDK
xmlrpc-common-3.1.3.jar Required by ESP SDK
xmlrpc-server-3.1.3.jar Required by ESP SDK
```

スキーマと設定ファイル

アダプタ設定はファイルからロードされ、アダプタのスキーマに対して検証されます。

例として提供されているフォルダには、サンプルのアダプタ設定ファイルがあります。

有効な設定ファイルを提供し、ファイルで指定されているアダプタ設定がアダプタ・スキーマに対して正しく検証されることを確認します。

アダプタ制御パラメータ

アダプタの **controllerPort** パラメータは、アダプタの「コマンドと制御」のポートを指定します。

このパラメータは、`config` ディレクトリにある `controller.xsd` ファイル内で定義されます。

パラメータ名	タイプ	説明
controllerPort	`positive integer`	(必須) アダプタの「コマンドと制御」のポートを指定。 ユーザ・コマンドは、localhost 上のこのポートに送信されます。

Event Stream Processor のパラメータ

Event Stream Processor のパラメータは、Event Stream Processor と TIBCO Rendezvous アダプタとの間の通信を設定します。

これらのパラメータは、`controller.xsd` ファイル (`config` ディレクトリ内) で定義されます。

パラメータ名	タイプ	説明
espAuthType	`string`	(必須) Event Stream Processor への認証に使用される方法を指定。 有効な値は、以下のとおりです。 • **server_rsa –** キーストアを使用する RSA 認証 • **user_password –** Kerberos 認証と LDAP 認証 • **none –** 認証なし アダプタがスタジオ・プラグインとして動作している場合、**espAuthType** は **Authentication Mode** スタジオ起動パラメータで上書きされます。
espUser	`string`	(必須) Event Stream Processor にログインするために必要なユーザ名を指定。 none (**espAuthType** を参照してください) 以外のすべての認証スキームで必要です。 デフォルト値はありません。
espPassword	`string`	(必須) Event Stream Processor にログインするために必要なパスワードを指定。 none (**espAuthType** を参照してください) 以外のすべての認証スキームで必要です。 **espPassword** 値が暗号化されているかどうかを示す "encrypted" 属性も指定します。デフォルト値は false です。 true に設定すると、パスワード値は **espRSAKeyStore** と **espRSAKeyStorePassword** を使用して復号化されます。

パラメータ名	タイプ	説明
espProjectUri	`string`	(必須) Event Stream Processor クラスタに接続するための完全なプロジェクト URI を指定。たとえば、`esp://localhost:19011/ws1/p1` のように記述します。
pulseInterval	`non-negative integer`	(必須) 期間を秒単位で指定。この期間では、アウトバウンド・レコードの変更は Event Stream Processor によって結合され、アダプタによって単一のイベントとして受信されるようになります。 設定されていないか、0 に設定されると、レコードの変更は、発生するごとに個々に受信されます。
espHeartbeatPeriod	`positive integer`	(オプション) 秒数を指定。この秒数が経過すると、アダプタは次のハートビートを Event Stream Processor に送信します。 Event Stream Processor が連続して 2 回のハートビート受信に失敗すると、アダプタがパブリッシュしたすべてのレコードが失効とマーク付けされます。 デフォルト値は 10 です。
recordQueueCapacity	`positive integer`	(オプション) レコード・キューの容量を指定。デフォルト値は、4096 です。
maxPubPoolSize	`positive integer`	(オプション) レコード・プールの最大サイズを指定。 レコード・プーリングは、パブリッシュ処理の高速化に役立ちます。デフォルト値は 256 です。
maxPubPoolTime	`positive integer`	(オプション) レコードがパブリッシュされるまでにプールに存在可能な最大時間をミリ秒単位で指定。 設定されていない場合、プーリング時間は無制限で、プーリング方式は **maxPubPoolSize** によって制御されます。 デフォルト値はありません。

パラメータ名	タイプ	説明
useTransactions	`boolean`	(オプション) true に設定すると、プールされたメッセージがトランザクションで Event Stream Processor にパブリッシュされる。 false に設定すると、エンベロープでパブリッシュされます。デフォルト値は false です。
espRSAKeyStore	`string`	(場合に応じて必須) RSA キーストアのロケーションを指定し、パスワード値を復号化するために使用される。 **espAuthType** が server_rsa に設定されている、**espPassword** の暗号化属性が true に設定されている、または両方の場合に必要です。
espRSAKeyStorePassword	`string`	(場合に応じて必須) キーストア・パスワードを指定し、パスワード値を復号化するのに使用される。**espAuthType** が server_rsa に設定されている、**espPassword** の暗号化属性が true に設定されている、または両方の場合に必要です。
espEncryptionAlgorithm	`string`	(オプション) **espPassword** の暗号化属性が true に設定されると使用される。 ブランクのままにすると、RSA がデフォルトとして使用されます。

ストリーム設定

設定ファイルの streams セクションを使用して、メッセージ・サブジェクトをデータ・ストリームにマップします。

入力ストリーム・パラメータ

inboundStreamType パラメータ内で定義される **name** パラメータと **subjects** パラメータは、データ・ストリームとメッセージ・サブジェクトの名前を指定します。

これらのパラメータは、`config` フォルダにある `tibcorvadapter.xsd` ファイル内で定義されます。

プロパティ ID	タイプ	説明
name	`string`	(必須) データ・ストリームの名前を指定。

プロパティ ID	タイプ	説明
subjects	`subjectsType`	(必須) 0 以上のメッセージ・サブジェクトを指定。 アダプタは起動時に、これらのサブジェクトのそれぞれにサブスクライブします。 ワイルドカードを使用したサブスクリプションがサポートされています。 指定したサブジェクトのいずれかに到着したインバウンド・メッセージは、このデータ・ストリームにパブリッシュされます。 複数のデータ・ストリームで、同じサブジェクトを指定できます。

出力ストリーム・パラメータ

出力ストリーム・パラメータは、**outboundStreamType** パラメータ内で定義されます。

これらのパラメータは、`tibcorvadapter.xsd` ファイル (`config` フォルダ内) で定義されます。

パラメータ ID	タイプ	説明
name	`string`	(必須) データ・ストリームの名前を指定。
gdmode	`boolean`	(詳細) true に設定すると、アダプタが、保証された配信 (GD) モードで動作し、GD 関連のすべてのパラメータが必須になる。デフォルト値は false です。
gdkeycolumnname	`string`	(詳細) GD キーを保持するフレックス演算子のカラム名を指定する。 GD キーは、対象となるストリームに格納される opcode に関係なく、各イベントを一意に識別する、常に増加する値です。 デフォルト値はありません。
gdopcodecolumnname	`string`	(詳細) opcode を保持するフレックス演算子のカラム名を指定する。 opcode は、対象のストリームで発生するイベントのオペレーション・コード (たとえば、挿入、更新、または削除) です。 デフォルト値はありません。
gdcontrolstream	`string`	(詳細) GD セットアップの制御ウィンドウの名前を指定する。 制御ウィンドウはソース・ストリームで、そのデータがアダプタによって処理済みで、安全に削除できることをフレックス演算子に通知します。 デフォルト値はありません。

パラメータ ID	タイプ	説明
gdbatchsize	`integer`	(詳細) レコード数を指定し、この数に到達すると、制御ウィンドウは最新の GD キーで更新される。デフォルト値は 1000 です。
gdpurgeinterval	`integer`	(詳細) レコード数を指定し、この数に到達すると、フレックス演算子はパージを実行する。デフォルト値は 1000 です。

参照：
- *第 5 章、「保証された配信」* (591 ページ)

Rendezvous サーバ設定
Rendezvous サーバ設定を、**rvDaemon**、**rvService**、**rvNetwork**、**cmmode**、**cmname**、**cmledgerfile** のパラメータを使用して設定します。

これらのパラメータは、`tibcorvadapter.xsd` ファイル (`config` フォルダ内) で定義されます。

パラメータ名	タイプ	説明
rvDaemon	`string`	(必須) Rendezvous サーバ・デーモンが実行するホスト名 (IP アドレス) とポート番号をコロンで区切って指定。
rvService	`string`	(オプション) Rendezvous サービスの名前を指定。
rvNetwork	`string`	(オプション) Rendezvous ネットワークの名前を指定。
cmmode	`boolean`	(必須) 入力ストリームと出力ストリームの両方に対して Certified Message (CM) 転送モードを有効化。デフォルト値は false です。
cmname	`string`	(オプション) CM 転送を他の CM 転送から区別する再使用可能な名前。 CM モードで値が指定されていない場合、デフォルトの 'sesp' が使用されます。この値は、**cmmode** が true に設定されている場合のみ考慮されます。
cmledgerfile	`string`	(オプション) CM 転送をファイル・ベースの元帳モードで実行するための有効なファイル・ロケーションを指定。 指定されていない場合、CM 転送はプロセス・ベースの元帳モードで実行します。 この値は、**cmmode** が true に設定されている場合のみ考慮されます。

TIBCO Rendezvous 設定ファイルのサンプル

TIBCO Rendezvous アダプタ用のサンプル設定ファイル (`adapter.xml`) を以下に示します。

このファイルは、`example` にあります。

```
<adapter>

<!-- Adapter controller -->
 <controller>
  <controllerPort>13579</controllerPort>
 </controller>

<!-- Sybase Stream processor settings -->
 <esp>
   <espConnection>
        <espProjectUri>esp://localhost:19011/w1/p1</espProjectUri>
   </espConnection>

   <espSecurity>
        <espUser>espuser</espUser>
        <espPassword encrypted="false">espuser</espPassword>
        <espAuthType>none</espAuthType>
<!--
        <espRSAKeyStore>/keystore/keystore.jks</espRSAKeyStore>
       <espRSAKeyStorePassword>Sybase123</espRSAKeyStorePassword> --
>
        <espEncryptionAlgorithm>RSA</espEncryptionAlgorithm>
   </espSecurity>
  <maxPubPoolSize>1</maxPubPoolSize>
 </esp>

<!-- Stream to subject mapping -->
 <streams>
  <inbound>
   <stream>
    <name>MyInStream</name>
    <subjects>
     <subject>MySubject</subject>
    </subjects>
   </stream>
  </inbound>
  <outbound>
   <stream>
    <name>MyOutStream</name>
        <gdmode>false</gdmode>
        <gdkeycolumnname>gdkey</gdkeycolumnname>
        <gdopcodecolumnname>gdopcode</gdopcodecolumnname>
        <gdcontrolstream>W1_truncate</gdcontrolstream>
        <gdbatchsize>2</gdbatchsize>
        <gdpurgeinterval>4</gdpurgeinterval>
   </stream>
  </outbound>
```

```
 </streams>

<!-- Rendezvous settings -->
 <rvSettings>
  <rvDaemon>localhost:7500</rvDaemon>
  <cmmode>false</cmmode>
  <cmname>sesp</cmname>
  <cmledgerfile>C:¥ledger.txt</cmledgerfile>
 </rvSettings>

</adapter>
```

Tibco Rendezvous アダプタ

TIBCO Rendezvous アダプタは、Rendezvous サブジェクトにストリーム・データをパブリッシュし、Rendezvous サブジェクトからのストリーム・データをパブリッシュします。

認証方法は、次の 3 つの中から Event Stream Processor と同じに設定されます。none、rsa、または gssapi。

このアダプタを使用するには、TIBCO Rendezvous アダプタのバージョン 1.0 以降をインストールします。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Connector Directory Path	**baseDir**	`directory`	(必須) アダプタ・インストール・ディレクトリへのパスを指定。このプロパティは、「コネクタ・リモート・ディレクトリ・パス」プロパティが指定されていると無視されます。 デフォルト値はありません。
Configuration File Path	**configFilePath**	`configFilename`	(必須) アダプタ設定ファイルの絶対パスを指定。 このプロパティは、「リモート設定ファイル・パス」プロパティが指定されていると無視されます。 デフォルト値はありません。

プロパティ・ラベル	プロパティ ID	タイプ	説明
Connector Remote Directory Path	**remoteBaseDir**	`string`	(詳細) アダプタ・リモート・ベース・ディレクトリのパスを指定(リモート実行のみ)。 このプロパティが指定されている場合、**Connector Directory Path** プロパティは無視されます。 デフォルト値はありません。
Remote Configuration File Path	**remoteConfigFilePath**	`string`	(詳細) アダプタ・リモート設定ファイルの絶対パスを指定 (リモート実行のみ)。 このプロパティが指定されている場合、**Configuration File Path** プロパティは無視されます。 デフォルト値はありません。
PropertySet	**propertyset**	`string`	(詳細) プロジェクト設定ファイルから使用するプロパティ・セット (プロパティと値から成るグループ) の名前を指定する。 プロジェクト設定ファイルと **ATTACH ADAPTER** 文で同じプロパティを指定すると、プロパティ・セットの値は、**ATTACH ADAPTER** 文で定義した値に優先します。 デフォルト値はありません。

ロギング

TIBCO Rendezvous アダプタは Apache `log4j` API を使用して、エラー、警告、情報、デバッグ・メッセージをログ記録します。

`log4j.properties` ファイルには、ログ記録設定があります。 サンプル `log4j.properties` ファイルが、アダプタ配布と共に提供されます。

ログ記録レベルを DEBUG に設定すると、ログ・ファイルが非常に大きくなることがあります。 デフォルトのレベルは、INFO です。 ログ記録設定の詳細については、http://logging.apache.org/log4j を参照してください。

オペレーション

アダプタをコマンド・ラインから操作します。

rvDaemon パラメータと、オプションの **rvService** パラメータと **rvNetwork** パラメータの設定が、Rendezvous サーバの設定と一貫していることを確認します。

イベントを処理するプロジェクトに、インバウンド・ストリームとアウトバウンド・ストリームが構成されていることを確認します。 該当するロギング・レベルを `log4j.properties` ファイルで設定します。

TIBCO Rendezvous アダプタの起動

TIBCO アダプタをコマンド・ラインから起動するには、Event Stream Processor を起動して、**start** コマンドを実行します。

1. Event Stream Processor を起動します。

   Windows：

   1. サンプル・クラスタを起動します。

      ```
      cd %ESP_HOME%¥cluster¥nodes¥node1
        %ESP_HOME%¥bin¥esp_server --cluster-node node1.xml
      ```

   2. CCL をコンパイルして CCX を作成します。

      ```
      %ESP_HOME%¥bin¥esp_compiler -i model.ccl -o model.ccx
      ```

   3. プロジェクトをクラスタに展開します。

      ```
      %ESP_HOME%¥bin¥esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --add_project --workspace-
      name=w1 --project-name=p1 --ccx=model.ccx
      ```

   4. 展開したクラスタをサーバで起動します。

      ```
      %ESP_HOME%¥bin¥esp_cluster_admin" --uri=esp://localhost:19011
      --username=sybase --password=sybase --start_project --
      workspace-name=w1 --project-name=p1
      ```

   UNIX：

   1. サンプル・クラスタを起動します。

      ```
      cd $ESP_HOME/cluster/nodes/node1
        $ESP_HOME/bin/esp_server --cluster-node node1.xml
      ```

   2. CCL をコンパイルして CCX を作成します。

      ```
      $ESP_HOME/bin/esp_compiler -i model.ccl -o model.ccx
      ```

   3. プロジェクトをクラスタに展開します。

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --add_project --workspace-
name=w1 --project-name=p1 --ccx=model.ccx
```

4. 展開したクラスタをサーバで起動します。

```
$ESP_HOME/bin/esp_cluster_admin" --uri=esp://localhost:19011
--username=sybase --password=sybase --start_project --
workspace-name=w1 --project-name=p1
```

2. アダプタを起動します。

オペレーティング・システム	手順

UNIX	端末ウィンドウを開き、次を入力。
	`cd $ESP_HOME/adapters/tibco_rv/bin`
	`./adapter.sh <configuration file path> start`

Windows	コマンド・ウィンドウを開き、次を入力。
	`cd %ESP_HOME%¥adapters¥tibco_rv¥bin`
	`adapter.bat <configuration file path> start`

注意： **esp_subscribe** ユーティリティを使用して、Rendezvous メッセージが正しく Event Stream Processor にパブリッシュされることを確認します。

参照：

- *start コマンド* (529 ページ)

TIBCO Rendezvous アダプタのステータスの確認

TIBCO アダプタのステータスをコマンド・ラインから確認するには、**status** コマンドを実行します。

オペレーティング・システム	手順

UNIX	端末ウィンドウを開き、次を入力。
	`cd $ESP_HOME/adapters/tibco_rv/bin`
	`./adapter.sh <configuration file path> status`

オペレーティング・システム	手順

Windows	コマンド・ウィンドウを開き、次を入力。
	`cd %ESP_HOME%/adapters/tibco_rv/bin`
	`adapter.bat <configuration file path> status`

アダプタのステータスが、実行中または停止中のいずれかとして表示されます。

参照：
- *status コマンド* (530 ページ)

TIBCO Rendezvous アダプタの停止
TIBCO アダプタをコマンド・ラインから停止するには、**stop** コマンドを実行します。

オペレーティング・システム	手順

UNIX	端末ウィンドウを開き、次を入力。
	`cd $ESP_HOME/adapters/tibco_rv/bin`
	`./adapter.sh <configuration file path> stop`

Windows	コマンド・ウィンドウを開き、次を入力。
	`cd %ESP_HOME%/adapters/tibco_rv/bin`
	`adapter.bat <configuration file path> stop`

参照：
- *stop コマンド* (530 ページ)

例： サブスクライブとパブリッシュ
サブジェクトにサブスクライブし、そのサブジェクトのアウトバウンド・レコードを Event Stream Processor にアップロードして、メッセージを Rendezvous サーバに送信します。次に、それをインバウンド・レコードとして受信し、パブリッシュします。

前提条件
TIBCO Rendezvous サーバの実行中のインスタンスとネットワーク接続している必要があります。 次の例をコマンド・ラインから操作します。

手順

1. `adapter.sh` スクリプトを編集します。
2. JAVA_HOME 環境変数を、Java Runtime Environment (JRE) がインストールされているディレクトリに設定します。
3. Event Stream Processor を起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開く。 1. サンプル・クラスタを起動。 `start_server_cluster.sh` 2. プロジェクトをクラスタ上で起動。 `start_project.sh`

Windows	コマンド・ウィンドウを開く。
	1. サンプル・クラスタを起動。
	`start_server_cluster.bat`
	2. プロジェクトをクラスタに追加し、クラスタ上で起動。
	`start_project.bat`

4. アダプタを起動します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./adapter.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `adapter.bat`

5. アダプタの初期化が完了するまで 5 ～ 10 秒待機します。
6. アウトバウンド・レコードをアップロードします。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./upload.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `upload.bat`

7. Event Stream Processor サブスクライバを起動して、データ・ストリームの内容を表示します。

オペレーティング・システム	手順
UNIX	端末ウィンドウを開き、次を入力。 `./esp-subscribe.sh`
Windows	コマンド・ウィンドウを開き、次を入力。 `esp-subscribe.bat`

8. Event Stream Processor にパブリッシュされたレコードを書き留めます。

第 3 章　カスタム・アダプタ

Event Stream Processor 内部アダプタのフレームワークを使用してカスタム内部アダプタを作成し、ESP が提供する SDK を使用してカスタム外部アダプタを作成します。

参照：
- *アダプタ・パラメータのデータ型* (9 ページ)

カスタム内部アダプタ

同梱されているアダプタによって要件が満たされない環境では、Event Stream Processor が提供する内部アダプタ・フレームワークを使用して、内部アダプタを作成できます。

カスタム内部アダプタを作成するには、カスタム・アダプタ・ライブラリを指定します。カスタム・アダプタ・ライブラリは、Event Stream Processor 共有ユーティリティ・ライブラリを使用して、外部データをサーバの形式に変換します。

アダプタ共有ユーティリティ・ライブラリは、カスタム・アダプタのライフ・サイクル関数と情報管理関数の実装を可能にする C インタフェースで API を公開します。C インタフェースによって、コンパイラに制限されることなく C または C++ でカスタム・アダプタを実装できます。

ヘッダ・ファイル `GenericAdapterInterface.h` では、アダプタ共有ユーティリティ・ライブラリでの関数のインポート宣言を指定します。

参照：
- *カスタム・アダプタ* (3 ページ)
- *カスタム外部アダプタ* (562 ページ)

アダプタ共有ユーティリティ・ライブラリ

アダプタ共有ユーティリティ・ライブラリによって、データ変換、データ操作、データ管理などのカスタム・アダプタの実装に必要なユーティリティ関数を使用できます。

ヘッダ・ファイル `GenericAdapterInterface.h` では、アダプタ共有ユーティリティ・ライブラリでのユーティリティ関数の呼び出しに必要なインポート宣言を指定します。

関数を呼び出すときに、各データ・ユーティリティには一意のハンドルが必要です。 アダプタ共有ユーティリティ・ライブラリは、Windows インストール環境では、`esp_adapter_util_lib.dll` とラベルが付けられていて、Linux インストール環境では、`libesp_adapter_util_lib.so` とラベルが付けられています。

アダプタ共有ユーティリティ・ライブラリで関数を呼び出すときに、各データ・ユーティリティには一意のハンドルが必要です。 たとえば、**CreateConnectionRow** を呼び出すことによって、**ConnectionRow** 関数を使用できます。 この呼び出しによって、void ポインタの形式で一意のハンドルが返されます。 **ConnectionRow** の下で他の API を呼び出すときに、このポインタを渡すことができます。

コールバック関数

アダプタの実装では、コールバック関数を使用して、サーバへのログオン、スキーマに関連した情報の取得、状態情報のサーバへの伝達を実行できます。

コールバック関数は、Windows インストール環境の場合 `esp_server_lib.dll` ファイル、Linux インストール環境の場合 `libesp_server_lib.dll` にあります。

`GenericAdapterInterface.h` ファイルでは、これらの関数のインポート宣言を指定します。

サンプル・モデル・ファイル

基本的なモデル・ファイルを作成するために使用できるサンプル構文。

このモデルは、文字列データの 2 つのカラムを使用するスキーマを表します。 また、サンプル・カスタム・アダプタの実装を参照する入力接続も定義します。

```
CREATE  MEMORY  STORE "memory" PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;

CREATE  INPUT  WINDOW Custom
SCHEMA (column1 STRING, column2 STRING)
PRIMARY KEY (column1)
STORE "memory";

ATTACH INPUT ADAPTER Connection1
TYPE custom_in
TO Custom;
```

アダプタ設定ファイル

内部アダプタ設定ファイルは、内部アダプタを開始、停止するために Event Stream Processor によって使用されるプロパティとコマンド、またスタジオからの内部ア

ダプタの設定を可能にするその他の情報が含まれる XML ファイル (`.cnxml`) です。

アダプタ設定ファイルによって、カスタム・アダプタ DLL ファイルの名前も作成されます。 アダプタをロードするときに、このライブラリ名が参照されます。

以下は、カスタム・アダプタ DLL ファイルの名前付けのサンプル・コードです。

```
<?xml version="1.0" encoding="UTF-8"?>
<Adapter type="input"
 id="custom_in"
 label="Custom Input"
 descr="Dummy Custom Input Adapter"
>
 <Library file="custom_in" type=binary"/>
 <Section></Section>
</Adapter>
```

注意： `.cnxml` ファイルは、`Adapter.xsd` ファイルに従います。

`.cnxml` ファイルとカスタム・アダプタ DLL ファイルを作成した後、それらのファイルを `ESP_HOME/lib/adapters` フォルダにコピーします。

アダプタ・ライフ・サイクル関数

すべてのアダプタは、一連のアダプタ・ライフ・サイクル・イベントの流れに従います。

API (application program interface)	説明
bool reset(void* adapters);	これは最初に呼び出すライフ・サイクル API。 この API を呼び出して入力アダプタと出力アダプタの両方を初期化します。
void start(void* adapter);	reset 呼び出しの直後にこの API を呼び出し、アダプタ実装に固有のデータ処理を行うために使用する。 入力アダプタと出力アダプタの両方に対して、この API を呼び出します。
void* getNext(void* adapter);	この API はデータを読み込み、サーバが認識する形式でそのデータへのポインタを返す。 アダプタ共有ユーティリティ・ライブラリは、データ変換関数を提供します。 入力アダプタに対してのみ、この API を呼び出します。

API (application program interface)	説明
void putNext(void* adapter,void* stream);	この API は提供されたデータをサーバが認識する形式に変換し、指定されたアダプタ実装に従ってそのデータを書き込む。 アダプタ共有ユーティリティ・ライブラリは、データ変換関数を提供します。 出力アダプタに対してのみ、この API を呼び出します。
void stop(void* adapter);	この API を呼び出して、アダプタを停止する。 入力アダプタと出力アダプタの両方に対して、この API を呼び出します。
void cleanup(void* adapter);	アダプタが停止した後に、この API を呼び出してクリーンアップ・アクティビティを実行する。
void commitTransaction(void* adapter);	この API は、トランザクションが終了したことを、出力アダプタに通知する。 出力アダプタに対してのみ、この API を呼び出します。
void putStartSync(void* adapter);	この API はベース・データの送信が開始されたことをアダプタ実装に通知する。 出力アダプタに対してのみ、この API を呼び出します。
void putEndSync(void* adapter);	この API はベース・データの送信が完了したことをアダプタ実装に通知する。 出力アダプタに対してのみ、この API を呼び出します。
void purgePending)void* adapter);	この API は出力アダプタに保留中のデータを消去するように指示する。 出力アダプタに対してのみ、この API を呼び出します。
bool isOutBase(void* adapter);	この API を呼び出して、アダプタがストリームのベースとなる内容を受信するかどうかを決定する。 出力アダプタに対してのみ、この API を呼び出します。

アダプタ・セットアップ関数

サーバは情報管理関数を使用して、アダプタの実装プロセスを完了します。

これらの関数は入力アダプタと出力アダプタの両方で使用されます。

API	説明
void* createAdapter();	これはアダプタを作成するときに、サーバが実行する最初の呼び出し。 関数は、サーバがアダプタへの後続の呼び出しを実行するために使用する一意のハンドルを返します。
void setCallBackReference(void*adapter,void* connectionCallBackReference);	サーバはこの API を呼び出して、サーバ側で **connectionCallBack** オブジェクトに対応する一意のハンドルをアダプタの実装に提供する。 サーバへのコールバックを実行するときに、このハンドルをパラメータとして使用します。
void setConnectionRowType(void* adapter,void* connectionRowType);	サーバはこの API を呼び出して、スキーマに関連する情報をアダプタの実装に提供する。
void setConnectionParams(void* adapter, void* connectionParams);	サーバはこの API を呼び出して、接続パラメータに関連する情報をアダプタの実装に提供する。

その他の関数

Event Stream Processor アダプタによってサポートされるその他の API について説明します。

API (application program interface)	説明
int getNumGood(void* adapter);	サーバはこの API を呼び出して、アダプタによって正常に処理されたローの数に関する情報をアダプタの実装から取得する。
int getNumBad(void* adapter);	サーバはこの API を呼び出して、アダプタによって正常に処理されなかったローの数に関する情報をアダプタの実装から取得する。
int getNumRows(void* adapter);	サーバはこの API を呼び出して、アダプタによって処理されたローの総数に関する情報をアダプタの実装から取得する。
bool canDiscover(void* adapter);	サーバはこの API を呼び出して、スキーマ検出機能をサポートするかどうかに関する情報をアダプタの実装から取得する。
bool hasError(void* adapter);	サーバはこの API を呼び出して、データの処理中にエラーが発生したかどうかに関する情報をアダプタの実装から取得する。
void getError(void* adapter, charerrorString);**	サーバはこの API を呼び出して、エラー情報をアダプタの実装から取得する。

アダプタ実行状態

アダプタは、Event Stream Processor と対話するときに、一連の実行状態 (RS) を経て進行します。

- RS_READY – アダプタが起動する準備が完了していることを示す。
- RS_INITIAL – アダプタが起動と初期ロードを実行していることを示す。 アダプタは、reset 関数が呼び出されると、RS_INITIAL 状態に入ります。
- RS_CONTINUOUS – アダプタが継続的に追加データを待機していることを示す。 RS_CONTINUOUS 状態が reset メソッドで設定された場合、入力アダプタは処理するデータと共に reset から戻り、出力アダプタはデータを受け入れる準備が整った状態で reset から戻ります。
- RS_IDLE – アダプタがタイムアウトしたか、切断されたソケット接続を再確立しようとしていることを示す。
- RS_DONE – アダプタがこれ以上データを返さないこと、またはポーリング間隔の経過後にデータを取得できないことを示す。
- RS_DEAD – アダプタが終了状態に入ったことを示す。 アダプタは restart 関数が呼び出されるまで動作しません。

ポーリングが有効な場合、入力アダプタの状態が、RS_CONTINUOUS から RS_IDLE に遷移することがあります。 一定の時間が経過した後、アダプタの状態を RS_CONTINUOUS に戻して、データ取得を再試行するようにします。

内部カスタム・アダプタのスキーマ検出

extern "C" 関数を使用して、カスタム内部アダプタでスキーマ検出を有効にできます。

メソッド	説明
extern "C" DLLEXPORT bool canDiscover(void* adapter)	アダプタが検出をサポートしているかどうかを確認するために、検出中にアダプタ・フレームワークによって最初に呼び出されるメソッド。検出をサポートするアダプタは true の値を返します。
extern "C" DLLEXPORT void setDiscovery(void* adapter)	このメソッドは検出モードで実行していることアダプタに通知する。
extern "C" DLLEXPORT int getTableNames(void* adapter, char* tables)**	このメソッドはテーブルに設定する文字列の配列へのポインタ。 このメソッドには、データベースでテーブルを検索する場合にはテーブル名、ディレクトリにあるファイルの特定のタイプを検索する場合にはファイル名が指定されます。

メソッド	説明
extern "C" DLLEXPORT int getFieldNames(void* adapter, char* names, const char* tableName)**	このメソッドはフィールド名を返す。
extern "C" DLLEXPORT int getFieldTypes(void* adapter, char* types, const char* tableName)**	このメソッドはフィールド・タイプを返す。
extern "C" DLLEXPORT int getSampleRow(void* adapter, char* row, const char* tableName, int pos)**	このメソッドはサンプル・ローを返す。

サンプル・カスタム内部アダプタの実装

カスタム内部アダプタの実装を作成するために使用できるサンプル構文。この実装には、カスタム・アダプタでのスキーマ検出を可能にする extern "C" メソッドが組み込まれます。

```
/*
 * CustomAdapterInterface.cpp
 *
 *      Author: sample
 */

#include "GenericAdapterInterface.h"
#include <vector>
#include <sstream>
#include <iostream>
#include <string>

using namespace std;

struct InputAdapter
{
    InputAdapter();
    void* connectionCallBackReference;
    void* schemaInfomartion;
    void* parameters;
    void* rowBuf;
    void* errorObjIdentifier;
    int _badRows;
    int _goodRows;
    int _totalRows;
    int getColumnCount();
    void setState(int st);
    bool discoverTables();
```

```
    bool discover(string tableName);
    vector<string> _discoveredTableNames;
    vector<string> _discoveredFieldNames;
    vector<string> _discoveredFieldTypes;
    vector<vector<string> > _discoveredRows;
    vector<string> _row1;
    vector<string> _row2;
    bool _discoveryMode;
};

InputAdapter::InputAdapter()
{
    rowBuf = NULL;
    _badRows = 0;
    _goodRows = 0;
    _totalRows = 0;
    _discoveryMode = false;
    _discoveredTableNames.clear();
    _discoveredFieldNames.clear();
    _discoveredFieldTypes.clear();
    _discoveredRows.clear();
    _row1.clear();
    _row2.clear();
}

int InputAdapter::getColumnCount()
{
    return ::getColumnCount(schemaInfomartion);
}

void InputAdapter::setState(int st)
{
    ::setAdapterState(connectionCallBackReference, st);
}

extern "C" DLLEXPORT
void* createAdapter()
{
    return new InputAdapter();
}

extern "C" DLLEXPORT
void setCallBackReference(void *adapter,void
*connectionCallBackReference)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    inputAdapterObject->connectionCallBackReference =
connectionCallBackReference;
}

extern "C" DLLEXPORT
void setConnectionRowType(void *adapter,void *connectionRowType)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    inputAdapterObject->schemaInfomartion = connectionRowType;
}
```

```
extern "C" DLLEXPORT
void  setConnectionParams(void* adapter,void* connectionParams)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    inputAdapterObject->parameters = connectionParams;
}

extern "C" DLLEXPORT
void* getNext(void *adapter)
{
    StreamRow streamRow = NULL;
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    int n = inputAdapterObject->getColumnCount();
    std::stringstream ss;
    if(inputAdapterObject->_totalRows <10){

        for (int column = 0; column < n; column++) {
            ss.str("");
            ss << inputAdapterObject->_totalRows;
            std::string tempString;
            tempString = ss.str();
            std::string row = "ROW";
            row.append(tempString);
            ss.str("");
            ss << column;
            tempString = ss.str();
            std::string columnString = "COLUMN";
            columnString.append(tempString);
            row.append(columnString);
            ::setFieldAsStringWithIndex(inputAdapterObject->rowBuf,
column, row.c_str());

        }

        inputAdapterObject->_totalRows++;
        streamRow = ::toRow(inputAdapterObject->rowBuf,
inputAdapterObject->_totalRows, inputAdapterObject-
>errorObjIdentifier);
        if( streamRow )
        {
            inputAdapterObject->_goodRows++;
        } else
        {
            inputAdapterObject->_badRows++;
        }


    } else {
        inputAdapterObject->setState(RS_DONE);
    }
    return streamRow;

}

extern "C" DLLEXPORT
```

```
bool reset(void *adapter)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    if(inputAdapterObject->rowBuf)
        delete inputAdapterObject->rowBuf;
    string type = "RowByOrder";
    inputAdapterObject->rowBuf
= ::createConnectionRow(type.c_str());
    ::setStreamType(inputAdapterObject->rowBuf, inputAdapterObject-
>schemaInfomartion, false);
    inputAdapterObject->errorObjIdentifier
=::createConnectionErrors();
    inputAdapterObject->setState(RS_CONTINUOUS);
    return true;
}

extern "C" DLLEXPORT
int getTotalRowsProcessed(void *adapter)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    return inputAdapterObject->_totalRows;
}

extern "C" DLLEXPORT
int getNumberOfBadRows(void *adapter)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    return inputAdapterObject->_badRows;
}

extern "C" DLLEXPORT
int getNumberOfGoodRows(void *adapter)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    return inputAdapterObject->_goodRows;
}

extern "C" DLLEXPORT
bool hasError(void *adapter)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    return !(::empty(inputAdapterObject->errorObjIdentifier));
}

extern "C" DLLEXPORT
void getError(void *adapter, char** errorString)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    ::getAdapterError(inputAdapterObject->errorObjIdentifier,
errorString);
}

extern "C" DLLEXPORT
void  start(void* adapter){}

extern "C" DLLEXPORT
```

```
void  stop(void* adapter){}

extern "C" DLLEXPORT
void  cleanup(void* adapter){}

extern "C" DLLEXPORT
bool  canDiscover(void* adapter){return true;}

extern "C" DLLEXPORT
void deleteAdapter(void* adapter)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    delete inputAdapterObject;
}

extern "C" DLLEXPORT
void commitTransaction(void *adapter){}



extern "C" DLLEXPORT
int getTableNames(void* adapter, char*** tables)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;

    if(!inputAdapterObject->discoverTables())
    {
        return 0;
    }

    (*tables) = (char**) malloc(sizeof(char*)*inputAdapterObject-
>_discoveredTableNames.size());

    for(int index=0; index < inputAdapterObject-
>_discoveredTableNames.size(); index++)
    {
        size_t tableNameSize = inputAdapterObject-
>_discoveredTableNames[index].size() + 1 ;
        char* tableName = new char [tableNameSize ];
        strncpy(tableName, inputAdapterObject-
>_discoveredTableNames[index].c_str(),tableNameSize);
        (*tables)[index] = tableName;
    }

    return inputAdapterObject->_discoveredTableNames.size();
}

extern "C" DLLEXPORT
int getFieldNames(void* adapter, char*** names, const char*
tableName)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;

    string table (tableName);

    if(!inputAdapterObject->discover(table))
```

```
    {
        return 0;
    }

    (*names) = (char**) malloc(sizeof(char*)*inputAdapterObject-
>_discoveredFieldNames.size());

    for(int index=0; index < inputAdapterObject-
>_discoveredFieldNames.size(); index++)
    {
        size_t fieldNameSize = inputAdapterObject-
>_discoveredFieldNames[index].size() + 1;
        char* fieldName = new char [ fieldNameSize ];
        strncpy(fieldName, inputAdapterObject-
>_discoveredFieldNames[index].c_str(),fieldNameSize);
        (*names)[index] = fieldName;
    }

    return inputAdapterObject->_discoveredFieldNames.size();

}

extern "C" DLLEXPORT
int getFieldTypes(void* adapter, char*** types, const char*
tableName)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;

    string table (tableName);

    if(!inputAdapterObject->discover(table))
    {
        return 0;
    }

    (*types) = (char**) malloc(sizeof(char*)*inputAdapterObject-
>_discoveredFieldTypes.size());


    for(int index=0; index < inputAdapterObject-
>_discoveredFieldTypes.size(); index++)
    {
        size_t fieldTypeSize = inputAdapterObject-
>_discoveredFieldTypes[index].size() + 1;
        char* fieldType = new char [ fieldTypeSize ];
        strncpy(fieldType, inputAdapterObject-
>_discoveredFieldTypes[index].c_str(), fieldTypeSize);
        (*types)[index] = fieldType;
    }

    return inputAdapterObject->_discoveredFieldTypes.size();
}

extern "C" DLLEXPORT
int getSampleRow(void* adapter, char*** row, const char* tableName,
int pos)
```

```
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;

    string table (tableName);

    if(!inputAdapterObject->discover(table))
    {
        return 0;
    }

    vector<string> vals;

    if (pos < (int)inputAdapterObject->_discoveredRows.size())
    {
        vals = inputAdapterObject->_discoveredRows[pos];

        (*row) = (char**) malloc(sizeof(char*)*vals.size());

        for(int index=0; index < vals.size(); index++)
        {
            size_t columnSize = vals[index].size() + 1;
            char* column = new char [ columnSize ];
            strncpy(column, vals[index].c_str(),columnSize);
            (*row)[index] = column;
        }
    }

    return vals.size();

}

extern "C" DLLEXPORT
void setDiscovery(void* adapter)
{
    InputAdapter *inputAdapterObject = (InputAdapter*)adapter;
    inputAdapterObject->_discoveryMode = true;
}

bool InputAdapter::discoverTables()
{
    _discoveredTableNames.push_back("Table1");
    _discoveredTableNames.push_back("Table2");
    _discoveredTableNames.push_back("Table3");
    _discoveredTableNames.push_back("Table4");
    _discoveredTableNames.push_back("Table5");
    return true;
}

bool InputAdapter::discover(string tableName)
{
    _discoveredFieldNames.clear();
    _discoveredFieldTypes.clear();
    _row1.clear();
    _row2.clear();
    _discoveredRows.clear();
    _discoveredFieldNames.push_back("Column1");
```

```
    _discoveredFieldNames.push_back("Column2");
    _discoveredFieldNames.push_back("Column3");
    _discoveredFieldNames.push_back("Column4");
    _discoveredFieldNames.push_back("Column5");
    _discoveredFieldTypes.push_back("integer");
    _discoveredFieldTypes.push_back("string");
    _discoveredFieldTypes.push_back("string");
    _discoveredFieldTypes.push_back("float");
    _discoveredFieldTypes.push_back("float");
    _row1.push_back("1");
    _row1.push_back("A");
    _row1.push_back("B");
    _row1.push_back("1.1");
    _row1.push_back("2.2");
    _row2.push_back("2");
    _row2.push_back("X");
    _row2.push_back("Y");
    _row2.push_back("3.3");
    _row2.push_back("4.4");
    _discoveredRows.push_back(_row1);
    _discoveredRows.push_back(_row2);
    return true;
}
```

カスタム外部アダプタ

Event Stream Processor によって提供されるセットにはない、アダプタを追加、作成するメカニズムが外部アダプタのフレームワークによって提供されます。

外部アダプタをフィールドに追加できます。外部アダプタによって、スタジオとサーバが外部のデータソースやシンクを管理し、これらと対話するために必要なすべての情報が提供されます。 これらのカスタム・アダプタは、Pub/Sub API を使用して作成されるデータソースやシンクの簡単なアプリケーションです。アダプタ設定ファイル (cnxml) をインストールしてから、組み込みアダプタと同じ方法で、外部アダプタをスタジオで使用できます。

参照：

- *カスタム・アダプタ* (3 ページ)
- *カスタム内部アダプタ* (549 ページ)
- *外部アダプタ* (136 ページ)

外部アダプタ設定ファイル

フレームワークによって、アダプタ・プロパティが構成される外部アダプタ設定ファイル (cnxml) の構造が定義されます。

外部アダプタ設定ファイルは XML ファイルで、外部アダプタの開始、停止、またはオプションでスキーマ検出を実行するために Event Stream Processor によって使

用されるプロパティとコマンド、またスタジオから外部アダプタを設定するために必要なその他の情報で構成されます。

以下の例では、Event Stream Processor に付属している 4 つのユーティリティ (**esp_convert**、**esp_upload**、**esp_client**、**esp_discxmlfiles**) を使用して、ファイルのディレクトリの参照、ソース・ストリームの作成、データのロードをサポートする、実際に動作する外部アダプタを完全に定義する cnxml ファイルが示されています。 サンプル設定ファイル (`simplified_xml_input_plugin.cnxml`) は、`$ESP_HOME/lib/adapters` ディレクトリにあります。 このディレクトリは、Event Stream Processor の標準配布パッケージにあります。

この例では、スクリプトの長い行は、読みやすさを考慮して、またフォーマットの問題を避けるために分割されています。 このサンプルを使用して独自の外部アダプタ設定ファイルを作成する場合、長さにかかわらず、すべてのコマンド・プロパティを 1 行で記述してください。

```
<?xml version="1.0" encoding="UTF-8"?>

<Adapter type="input" external="true"
  id="simplified_xml_input_plugin"
  label="Simplified external XML file input plugin Adapter"
  descr="Example of uploading an XML file through a simple external
Adapter"
>
  <Library file="simple_ext" type="binary"/>

  <!--
     The special section contains the special internal parameters
     which are prefixed with "x_". Although these are parameters,
     the framwork requires them to be defined using the <Internal
     .../> element. They are hidden from the user in ESP Studio.
  -->
  <Special>
    <Internal id="x_initialOnly"
      label="Does Initial Loading Only"
      descr="Do initial loading, or the continuous loading"
      type="boolean"
      default="true"
    />
    <Internal id="x_addParamFile"
      label="Add Parameter File"
      type="boolean"
      default="false"
    />
    <Internal id="x_killRetryPeriod"
      label="Period to repeat the stop command until the process
exits"
      type="int"
      default="1"
    />

    <!--
```

```
    Convert a file of xml record to ESP Binary format using
esp_convert;
        pipe into the esp_upload program, naming the upload
connection:
    $platformStream.$platformConnection
    -->
    <Internal id="x_unixCmdExec"
      label="Execute Command"
      type="string"
      default="$ESP_HOME/bin/esp_convert -p $platformCommandPort
&amp;lt;&amp;quot;$directory/$filename&amp;quot; | $ESP_HOME/bin/esp_upload -m
$platformStream.$platformConnection -p $platformCommandPort"
    />
    <Internal id="x_winCmdExec"
      label="Execute Command"
      type="string"
     default="$+/{$ESP_HOME/bin/esp_convert} -p $platformCommandPort
&amp;lt;&amp;quot;$directory/$filename&amp;quot; | $+/{$ESP_HOME/bin/esp_upload}
-m $platformStream.$platformConnection -p $platformCommandPort"
    />

    <!--
        use the esp_client command to stop an existing esp_upload
connection named:
        $platformStream.$platformConnection
    -->
    <Internal id="x_unixCmdStop"
      label="Stop Command"
      type="string"
     default="$ESP_HOME/bin/esp_client -p $platformCommandPort 'kill
every {$platformStream.$platformConnection}' &amp;lt;/dev/null"
    />
    <Internal id="x_winCmdStop"
      label="Stop Command"
      type="string"
     default="$+/{$ESP_HOME/bin/esp_client} -p $platformCommandPort
&amp;quot;kill every {$platformStream.$platformConnection}&amp;quot;
&amp;lt;nul"
    />

    <!--
    Use the esp_discxmlfiles command to do data discovery.
    The command below will have '-o "<temp file>"' added to it. It
    will write the discovered data in this file.
     -->
    <Internal id="x_unixCmdDisc"
      label="Discovery Command"
      type="string"
      default="$ESP_HOME/bin/esp_discxmlfiles -d &amp;quot;
$directory&amp;quot;"
    />
    <Internal id="x_winCmdDisc"
      label="Discovery Command"
      type="string"
      default="$+/{$ESP_HOME/bin/esp_discxmlfiles} -d &amp;quot;$+/
{$directory}&amp;quot;"
```

```
    />
  </Special>

  <Section>

    <!--
       Any parameter defined here, is visible in the ESP Studio, and
may
    be configured by the user at runtime in the data location
explorer.
    These are defined according to the $ESP_HOME/etc/Adapter.xsd
    schema.
    -->

    <Parameter id="filename"
      label="File"
      descr="File to upload"
      type="tables"
      use="required"
    />
    <Parameter id="directory"
      label="path to file"
      descr="directory to search"
      type="directory"
      use="required"
    />
    <Parameter id="propertyset"
      label="propertyset"
      descr="to look up properties in project configuration"
      type="string"
      use="advanced"
      default=""/>
  </Section>
</Adapter>
```

外部アダプタ・プロパティ

サンプル cnxml ファイルである `$ESP_HOME/lib/connections/PLUGIN_TEMPLATE.cnxml` の例を参照してください。このファイルはコピーしてカスタマイズできます。 このファイルには、使用可能なすべての内部パラメータが埋め込まれており、使用方法を示すコメント・ブロックもあります。

プロパティ ID	タイプ	説明
x_paramFile	`string`	アダプタ・フレームワークがすべての内部パラメータとユーザ定義パラメータを記述するファイル名を指定する。ファイル名を指定することによって、他の内部パラメータを使用できます。 次に例を示します。 `/tmp/mymodel.$platformStream.$platformConnector.$platformCommandPort.cfg`

プロパティ ID	タイプ	説明
x_paramFormat	`string`	`prop`、`shell`、または `xml` に設定して、パラメータ・ファイルのフォーマットを選択する。
x_addParamFile	`boolean`	パラメータ・ファイル名をすべての `x_cmd*` 文字列に自動的に付加するかどうかを決定する。 たとえば、`cmd -f` としてコマンドを指定する場合、このプロパティが true に設定されていると、実際のコマンドは `cmd -f <value of x_paramFile>` として実行されます。
x_initialOnly	`boolean`	true の場合、初期ロードのみを実行する。 継続してロードする場合は false に設定します。 初期ロードは、開始して、一部の静的データをロードし、終了するアダプタに役立ち、これによって、別のアダプタ・グループを段階的ロード・シナリオで起動させることができます。
x_killRetryPeriod	`integer`	このパラメータが `>0` である場合、**x_{unix,win}CmdExec** コマンドが返されたことをフレームワークが検出するまで、**x_killRetry** 秒ごとに **x_{unix,win}CmdStop** コマンドがリトライされる。 このパラメータがゼロの場合、**x_{unix,win}CmdStop** コマンドは 1 回だけ実行され、**x_{unix,win}CmdExec** コマンドが停止すると想定されています。

外部アダプタ・コマンド

外部アダプタ・コマンドは 2 種類のカテゴリーに分かれます。 1 つの種類はスタジオと同じホストで実行されるコマンドで、もう 1 つの種類はサーバと同じホストで実行されるコマンドです。

検出コマンド (**x_unixDiscCmd** と **x_winDiscCmd**) は、常にスタジオ・ホストで実行されます。 その他すべてのコマンドは、サーバ・ホストで実行されます。

スタジオとサーバが同じホストで実行されることが多いため、カスタム・アダプタのすべてのコマンドと駆動スクリプトを簡単に開発できます。スタジオとサーバが異なるホストで実行されている場合は、リモート実行の設定がより複雑になります。

たとえば、スタジオが Windows ホストで実行され、サーバがスタジオを介してリモートの Linux ホストで実行するように設定されている場合、フレームワークが生成する検出コマンドと検出ファイル名は、Windows 環境で実行、生成されることになります。検出ファイルへのパスは Windows 固有のパスとなり、ドライブ文字とパス区切り文字として '¥' 文字が使用されます。 この場合、コネクタの開発者

は、検出コマンドを Windows 環境で実行するように記述する必要がありますが、その他のすべてのコマンドはユーザ設定の **ssh** または **rsh** コマンドを使用して Linux ボックスでリモート実行されるようにコーディングする必要があります。

コマンド	説明
x_unixCmdConfig **x_winCmdConfig**	設定コマンドでは、パラメータの必要な解析か検査、またはその両方を実行する必要がある。 パラメータ・ファイルの読み込み、解析、書き直しによって、実行コマンドが予期する実際のフォーマットにパラメータを変換する場合もあります。 設定コマンドが失敗する (0 以外の値を返す) 場合、`reset()` エラーとしてレポートされ、アダプタは起動に失敗します。
x_unixCmdExec **x_winCmdExec**	サーバがアダプタを起動すると、このコマンドが実行され、終了時にコネクタが終了したことが示される。
x_unixCmdStop **x_winCmdStop**	このコマンドは独立したスレッドから実行され、**x_{unix,win}CmdExec** コマンドによって作成されたすべてのプロセスを停止する。その結果、**x_{unix,win}CmdExec** が返されます。
x_unixCmdClean **x_winCmdClean**	このコマンドは、サーバが接続を停止した (つまり、**x_{unix,win}CmdExec** が返されたとき) 後に実行される。

x_winDiscCmd	このコマンドはスキーマ検出用。 コマンドに渡される名前のファイルに検出ファイルを書き込む必要があります。 パラメータ **-o <temporary disc filename>** の引数がこのコマンドに付加されてから、実行されます。
	`<discover>`
	`  <table name="table_name_1" />`
	`  <column name="col_name_1" type="col_type_1"/>`
	`  .`
	`  .`
	`  .`
	`  <column name="col_name_k" type="col_type_k"/>`
	`  </table>`
	`  .`
	`  .`
	`  .`
	`  <table name="table_name_n" />`
	`  <column name="col_name_1" type="col_type_1"/>`
	`  .`
	`  .`
	`  .`
	`  <column name="col_name_1" type="col_type_1"/>`
	`  </table>`
	`</discover>`

ユーザ定義パラメータとパラメータの代入

内部パラメータと任意の数のユーザ定義パラメータを cnxml ファイルに作成できます。

すべてのシステム・パラメータとユーザ定義パラメータは、コマンドまたはスクリプトの引数で参照できます。 これらのパラメータは、シェル代入変数と同様の方法で動作します。 最も簡単な例は、`simplified_xml_input_external.cnxml file` からの動作です。 以下の長い行の一部は、読みやすさを考慮して、またフォーマットの問題を避けるために分割されています。

```
    <Internal id="x_unixCmdExec"
      label="Execute Command"
      type="string"
      default="$ESP_HOME/bin/esp_convert
                        -p $platformCommandPort &lt;&quot;$directory/
$filename&quot; | $ESP_HOME/bin/esp_upload
                         -m $platformStream.$platformConnection
                         -p $platformCommandPort"
    />
```

ESP_HOME などの外部環境変数、内部システム・パラメータ (**platformCommandPort**)、ユーザ定義パラメータ (filename) は、拡張される場合があります。 パラメータ拡張の完全セマンティックは次のとおりです。

```
$name
${name}
${name=value?substitution[:substitution]}
${name<>value?substitution[:substitution]}
${name!=value?substitution[:substitution]}
${name==value?substitution[:substitution]}
${name<value?substitution[:substitution]}
${name<=value?substitution[:substitution]}
${name>value?substitution[:substitution]}
${name>=value?substitution[:substitution]}
```

`{}` を使用するすべての形式は、`$` の後に `+` が加えられる場合もあります (たとえば、`$+{name}`)。 `+` が加えられると、代入の結果が再度解析され、その解析された結果の値によって置き換えられます。 ¥ 記号は、次の文字をエスケープし、特殊な解釈がされないようにします。

条件式は、パラメータの値を定数値と比較して、成功した場合は最初の代入を使用し、失敗した場合は 2 番目の代入を使用します。 比較演算子 `==` と `!=` は、数値として値を比較します。 `=` 比較演算子と `<>` は、文字列として値を比較します。 `?`、`:`、`}` などの文字を値に指定する場合は、¥ を使用してエスケープする必要があります。 `{` と `}` の文字を代入値に指定する場合は、対になっている必要が

あります。対になっていないすべてのカッコは、¥ を使用してエスケープする必要があります。 引用符文字は、特殊文字として扱われません。

$+{...} の代入の形式には、他の変数への参照が指定される場合があります。 これは、もう一度代入を繰り返した代入の結果を渡すことによって実装されます。 その結果、エスケープするために余分に ¥ を使用する必要が生じます。 たとえば、文字列 $+{name=?¥¥¥¥} は、パラメータ **name** が空の場合、¥ を 1 つ生成します。 最初に渡されるとき、対になる円記号はそれぞれ 1 つの円記号に変えられますが、2 度目に渡されるときには、¥¥ は 1 つの円記号に変わります。

Windows 用の特殊な代入構文：

$/{value} $+/{value}	Windows のパス指定ですべてのスラッシュをエスケープする必要性を排除するために、値内のすべてのスラッシュを円記号で置き換える。
$%{value} $+%{value}	Windows でエスケープするために、すべての % を %% に置き換える。

実行のためにシェルまたは cmd.exe に文字列が結果として渡される場合、シェルまたは cmd.exe は独自の代入も実行します。

以下は、より強力な代入機能の一部を使用して、単純な例で実行コマンドを定義する例です。 ただし、オプションの認証や暗号化、ユーザ定義の日付フォーマットをサポートする条件機能を使用する場合があります。

```
<Internal id="x_unixCmdExec"
        label="Execute Command"
     type="string"
  default="$ESP_HOME/bin/esp_convert
           ${platformSsl==1?-e}
       $+{dateFormat<>?-m '$dateFormat'}
                -c '$+{user=?user:$user}:$password'
                -p $platformCommandPort
                   <"$directory/$filename" |
                   $ESP_HOME/bin/esp_upload
                   ${platformSsl==1?-e}
    -m $platformStream.$platformConnection
                -c '$user:$password'
    -p $platformCommandPort"
/>
```

自動生成されたパラメータ・ファイル

基本的な外部アダプタ・フレームワークは、起動時にパラメータ (システムとユーザ定義) のセットをパラメータ・ファイルに書き込みます。

このファイルは、次のいずれかで記述されます。

- Java プロパティ
- シェルの割り当て
- 単純な XML フォーマット

コマンドは、パラメータ・ファイルにフル・アクセスできます。

次に、`simplified_xml_input_plugin.cnxml` ファイルのパラメータの例を示します。

```
             <Internal id="x_paramFile"
               label="Parameter File"
               type="string"
               default="/tmp/PARAMETER_FILE.txt"
             />
          <Internal id="x_paramFormat"
            label="Parameter Format"
            type="string"
            default="prop"
        />
     <Internal id="x_addParamFile"
       label="Add Parameter File"
       type="boolean"
       default="false"
/>
```

パラメータ・ファイルは、`/tmp/PARAMETER_FILE.txt` に書き込まれます。

```
            directory=/home/sjk/work/aleri/cimarron
            /branches/3.1/examples/input/xml_tables
            filename=trades.xml
            platformAuth=none
            platformCommandPort=31415
            platformConnection=Connection1
            platformHost=sjk-laptop
            platformSqlPort=22200
            platformSsl=0
            platformStream=Trades
```

または、Java プロパティ・フォーマットで、パラメータの完全なリストに記述されます。 フォーマットは、シェルの割り当ての `shell`、または単純な XML フォーマットの `xml` として指定できます。

x_addParamFile を true として指定した場合、

```
            <Internal id="x_addParamFile"
                  label="Add Parameter File"
               type="boolean"
                  default="true"
                />
```

実行される前に、引数 `/tmp/PARAMETER_FILE.txt` がすべてのコマンドに追加されます。

configFilename パラメータ

configFilename パラメータによって、スタジオでユーザ編集可能な設定ファイルを指定できます。

ユーザ定義の **confgFilename** パラメータを作成する場合、スタジオでこのフィールドの値部分をクリックすると、ローカル・ファイル・システム上のファイルを選択できるファイル・セレクタ・ダイアログが表示されます。 読み取り専用の名前を右クリックすると、ファイルの内容を変更できる別のダイアログが表示されます。 このダイアログによって、ユーザ編集可能な設定ファイルを指定できます。

カスタム外部パラメータのデータ型

`adapter.xsd` スキーマは、ユーザ定義のパラメータ用のいくつかのデータ型をサポートします。

サポートされるデータ型については、「はじめに」の「アダプタ・パラメータのデータ型」を参照してください。

カスタム外部アダプタでは、以下のデータ型はサポートされません。

- `runtimeFilename`
- `runtimeDirectory`
- `text`
- `query`
- `permutation`

注意： **start** コマンドと **stop** コマンドはサーバによって実行され、検出はスタジオによって実行されます。 この違いが、これらのパラメータの使用に影響を及ぼします。

参照：

- *アダプタ・パラメータのデータ型* (9 ページ)

カスタム外部アダプタの作成

以下は、SDK を使用してカスタム・アダプタを作成する一般的な手順です。

1. SDK インスタンスを取得します。
2. 必要な認証タイプのクレデンシャルを作成します。
3. このクレデンシャルを使用してプロジェクトに接続します。
4. サーバにパブリッシュするパブリッシャを作成します。
5. サーバからレコードにサブスクライブするサブスクライバを作成します。

6. パブリッシュまたはサブスクライブします。

参照：
- *カスタム・アダプタ* (3 ページ)
- *Java 外部アダプタ* (572 ページ)
- *C/C++ 外部アダプタ* (576 ページ)
- *.Net 外部アダプタ* (579 ページ)

Java 外部アダプタ
Java SDK を使用して、カスタム Java 外部アダプタを作成します。

参照：
- *カスタム・アダプタ* (3 ページ)
- *C/C++ 外部アダプタ* (576 ページ)
- *.Net 外部アダプタ* (579 ページ)
- *カスタム外部アダプタの作成* (571 ページ)

プロジェクトへの接続
認証クレデンシャルを使用してプロジェクトに接続します。

1. プロジェクトを取得します。
```
String projectUriStr = "esp://localhost:19011/ws1/p1“;
Uri uri = new Uri.Builder(projectUriStr).create();
project = sdk.getProject(uri, credentials);
```
2. プロジェクトに接続します。
```
 project.connect(60000);
```
   ここで、`60000` は、タイムアウトするまでに、サーバが接続呼び出しの完了を待機するミリ秒単位の時間を表します。

パブリッシャの作成
パブリッシャを作成、接続し、メッセージをパブリッシュします。

1. パブリッシャを作成、接続します。
```
Publisher pub = project.createPublisher();
pub.connect();
```
2. メッセージを作成、パブリッシュするには、ストリームとストリーム名の呼び出し、メッセージ・ライタの呼び出し、ロー・ライタの呼び出しを実行し、パブリッシュします。
```
String streamName = “Stream1”;
Stream stream = project.getStream(streamName);
```

```
MessageWriter mw = pub.getMessageWriter(streamName);
RelativeRowWriter writer= mw.getRelativeRowWriter();
mw.startEnvelope(0);  // can also  be mw.startTransaction() for
transactions.
for (int i = 0; i < recordsToPublish.length; i++) {
    addRow(writer, incomingRecords[i], stream);
}
mw.endBlock();
pub.publish(mw);
```

addRow のサンプル Java コード

addRow 操作によって、1 つのレコード・ローが、サーバにパブリッシュされたメッセージに追加されます。

新規ローでテーブルを更新するために、opcode を使用します。

```
Schema schema = stream.getEffectiveSchema();
DataType[] colTypes = schema.getColumnTypes();
rowWriter.startRow();
rowWriter.setOperation(Stream.Operation.UPSERT);
for (int fieldIndex = 0; fieldIndex < schema.getColumnCount();
fieldIndex++) {
    String name = (String) colNames[fieldIndex];
    attValue = record.get(fieldIndex);
switch(dataType){
    case BOOLEAN:     writer.setBoolean((Boolean) attValue); break;
    case INTEGER:     writer.setInteger((Integer) attValue); break;
          case TIMESTAMP:     writer.setTimestamp((Date) attValue);
break;
    }//switch
}//for loop
rowWriter.endRow();
```

コールバックを使用したサブスクライブ

新しいデータのコールバックを実施します。

1. サブスクライバのオプションを作成します。

   ```
   SubscriberOptions.Builder builder = new
   SubscriberOptions.Builder();
   builder.setAccessMode(AccessMode.CALLBACK);
   builder.setPulseInterval(pulseInterval);
   SubscriberOptions opts = builder.create();
   ```

   アクセス・モードを [CALLBACK] に設定し、コールバックを実行する頻度に関するパルス間隔を設定します。

2. サブスクライバを作成し、コールバックを登録します。

   ```
   Subscriber sub = project.createSubscriber(opts);
       sub.setCallback(EnumSet.allOf(SubscriberEvent.Type.class),
   this);
       sub.subscribeStream(streamName);
       sub.connect();
   ```

sub.setCallback は、processEvent メソッドを実装し、コールバック・メカニズムによって呼び出されるクラスです。

3. サブスクライバに登録するため使用するコールバック・クラスを作成します。
   a) Callback<SubscriberEvent> を実装する。
   b) getName() メソッドと processEvent(SubsriberEvent) メソッドを実装する。

```
     public void processEvent(SubscriberEvent event) {
         switch (event.getType()) {
          case SYNC_START:    dataFromLogstore=true;    break;
            case SYNC_END:     dataFromLogStore=false;
break;              case ERROR:    handleError(event);
break;
           case DATA:         handleData(event);         break;
            case DISCONNECTED: cleanupExit();        break;
        }
    }
```

この例では、手順 4 で参照される handleData と呼ばれる別のメソッドが宣言されています。メソッドの名前は可変です。

注意： イベントを受信すると、コールバック・メカニズムは processEvent を呼び出し、イベントをこのメソッドに渡します。

4. (オプション) handleData を使用して、サブスクライブされたデータを取得、使用するための別のメソッドを完成します。 それ以外の場合、データを processEvent で直接処理できます。

```
public void handleData(SubscriberEvent event) {
       MessageReader reader = event.getMessageReader();
        String streamName= event.getStream().getName();
       while ( reader.hasNextRow() ) {
            RowReader row = reader.nextRowReader();
            int ops= row.getOperation().code();
           String[] colNames=row.getSchema().getColumnNames();
           List record = new ArrayList<Object>();
           for (int j = 0; index = 0; j <
row.getSchema().getColumnCount(); ++j) {
             if ( row.isNull(j)) { record.add(index,null);  index
++;  continue;  }
               switch ( row.getSchema().getColumnTypes()[j]) {
                     case BOOLEAN: record.add(j,
row.getBoolean(j));break;
                     case INTEGER: record.add(j,
row.getInteger(j));break;
                     case TIMESTAMP: record.add(j,
row.getTimestamp(j)); break;
                  }//switch
           }//for loop
           sendRecordToExternalDataSource(record);
       }//while loop
   }//handleData
```

handleData イベントにはメッセージ・リーダがあり、ストリーム名を取得します。また、サブスクライブされるデータがあると、ロー・リーダを使用して新規ローを検索します。 データ型は指定されます。

直接アクセス・モードを使用したサブスクライブ

直接アクセス・モードは、テスト目的のみに使用することをおすすめします。

```
Subscriber sub = p.createSubscriber(); sub.connect();
sub.subscribeStream(“stream1");
while (true) {
    SubscriberEvent event = sub.getNextEvent();
    handleEvent(event);
}
```

コールバックを使用したパブリッシュ

コールバック・モードでのパブリッシュは、特殊な状況で使用できますが、おすすめしません。

```
PublisherOptions.Builder builder = new PublisherOptions.Builder();
builder.setAccessMode(AccessMode.CALLBACK);
builder.setPulseInterval(pulseInterval);
PublisherOptions opts = builder.create();
    Publisher pub = project.createPublisher(opts);
    pub.setCallback(EnumSet.allOf(PublisherEvent.Type.class), new
PublisherHandler(project));
    pub.connect();
```

PublisherHandler は Callback<PublisherEvent> を実装します。 また、次の 2 つのメソッドを実装します。getName() および processEvent(PublisherEvent event)

processEvent を実装するスクリプトは、次のようになります。

```
public void processEvent(PublisherEvent event) {
     switch (event.getType()) {
         case CONNECTED:     mwriter =
event.getPublisher().getMessageWriter(mstr);
                 rowwriter = mwriter.getRelativeRowWriter(); break;
         case READY: mwriter.startTransaction(0);
             for (int j = 0; j < 100; ++j) {
                    mrowwriter.startRow();
                     mrowwriter.setOperation(Operation.INSERT);
                  for (int i = 0; i < mschema.getColumnCount(); ++i)
{
                switch (mtypes[i]) {
                 case INTEGER: mrowwriter.setInteger(int_value+
+);break;
              case DOUBLE: mrowwriter.setDouble(double_value+=1.0);
break;
                }
                      }//columns
                      mrowwriter.endRow();
```

```
            }//for
             event.getPublisher().publish(mwriter);
        case ERROR: break;
       case DISCONNECTD:break;
    }//switch
}//processEvent
```

C/C++ 外部アダプタ

C/C++ SDK を使用して、カスタム C/C++ 外部アダプタを作成します。

参照：

- *カスタム・アダプタ* (3 ページ)
- *Java 外部アダプタ* (572 ページ)
- *.Net 外部アダプタ* (579 ページ)
- *カスタム外部アダプタの作成* (571 ページ)

プロジェクトの取得

認証クレデンシャルを作成し、そのクレデンシャルを使用してプロジェクトを作成します。

すべての SDK の呼び出しは、外部 C 呼び出しとして使用できます。

1. 認証用のクレデンシャル・オブジェクトを作成します。

```
#include <sdk/esp_sdk.h>
#include <sdk/esp_credentials.h>
        EspError* error = esp_error_create();
         esp_sdk_start(error);
        EspCredentials * m_creds =
esp_credentials_create(ESP_CREDENTIALS_USER_PASSWORD, error);
        esp_credentials_set_user(espuser.c_str(),error);
        esp_credentials_set_password(m_creds,
esppass.c_str(),error);
```

2. プロジェクトを作成します。

```
 EspUri* m_espUri = NULL; EspProject* m_project = NULL;
    if ( isCluster){
        m_espUri = esp_uri_create_string(project_uri.c_str(),
error);
       m_project = esp_project_get(m_espUri, m_creds ,NULL,error);
        esp_project_connect (m_project,error);
```

パブリッシュとサブスクライブ

パブリッシャとサブスクライバを作成し、コールバック・インスタンスを実装します。

1. パブリッシャを作成します。

```
EspPublisherOptions* publisherOptions =
esp_publisher_options_create (error);
```

```
Int rc
EspPublisher * m_publisher = esp_project_create_publisher
(m_project,publisherOptions,error);
EspStream* m_stream = esp_project_get_stream (m_project,m_opts-
>target.c_str(),error);
rc = esp_publisher_connect (m_publisher,error);
```

2. パブリッシュします。

注意： この手順のサンプル・コードには、ローをメッセージに追加する構文があります。

```
EspMessageWriter* m_msgwriter   =   esp_publisher_get_writer
(m_publisher,m_stream,error);
 EspRelativeRowWriter* m_rowwriter =
esp_message_writer_get_relative_rowwriter(m_msgwriter, error);
    const EspSchema* m_schema = esp_stream_get_schema
(m_stream,error);
    int numColumns;
    rc = esp_schema_get_numcolumns (m_schema, &numColumns,error);
   rc = esp_message_writer_start_envelope(m_msgwriter, 0, error);
    rc = esp_relative_rowwriter_start_row(m_rowwriter, error);
   rc = esp_relative_rowwriter_set_operation(m_rowwriter, (const
ESP_OPERATION_T)opcode, error);
int32_t colType;
    for (int j = 0;j < numColumns;j++){
    rc = esp_schema_get_column_type (m_schema,j,&colType,error);
    switch (type){
        case ESP_DATATYPE_INTEGER:
            memcpy (&integer_val,(int32_t *)
(dataValue),sizeof(uint32_t));
            rc = esp_relative_rowwriter_set_integer(m_rowwriter,
integer_val, error);
            break;
        case ESP_DATATYPE_LONG:
            memcpy (&long_val,(int64_t *)
(dataValue),sizeof(int64_t));
            rc = esp_relative_rowwriter_set_long(m_rowwriter,
long_val, error);
            break;
        }
}//for
rc = esp_relative_rowwriter_end_row(m_rowwriter, error);
rc = esp_message_writer_end_block(m_msgwriter, error);
rc = esp_publisher_publish(m_publisher, m_msgwriter, error);
```

3. サブスクライバのオプションを作成します。

```
EspSubscriberOptions * m_subscriberOptions =
esp_subscriber_options_create (error);
    int rc = esp_subscriber_options_set_access_mode(options,
CALLBACK_ACCESS, m_error);
    EspSubscriber * m_subscriber = esp_project_create_subscriber
(m_project,m_subscriberOptions,error);
    rc = esp_subscriber_options_free(options, m_error);
    rc = esp_subscriber_set_callback(subscriber ,
```

```
ESP_SUBSCRIBER_EVENT_ALL,
            subscriber_callback, NULL, m_error);
     subscriber_callback is global function which will get called
up.
```

4. コールバックを使用してサブスクライブします。

```
void subscriber_callback(const EspSubscriberEvent * event, void *
data) {
    uint32_t type;
    rc = esp_subscriber_event_get_type(event, &type, error);
     switch (type) {
                case ESP_SUBSCRIBER_EVENT_CONNECTED:
init(event,error);break;
        case ESP_SUBSCRIBER_EVENT_SYNC_START:     fromLogStore =
true; break;
             case ESP_SUBSCRIBER_EVENT_SYNC_END:    fromLogStore
= false; break;
                case ESP_SUBSCRIBER_EVENT_DATA:
handleData(event,error); break;
                case ESP_SUBSCRIBER_EVENT_DISCONNECTED:
cleanupaExit(); break;
                case ESP_SUBSCRIBER_EVENT_ERROR:
handleError(event,error); break;
    }
}//end subscriber_callback
```

handleData

handleData メソッドのサンプル C/C++ コード。

```
 EspMessageReader * reader = esp_subscriber_event_get_reader(event,
error);
     EspStream * stream = esp_message_reader_get_stream(reader,
error);
    const EspSchema * schema = esp_stream_get_schema(stream, error);
     EspRowReader * row_reader;
    int32_t int_value;   int64_t long_value; time_t date_value;
double double_value;
     int numcolumns, numrows, type;
     rc = esp_schema_get_numcolumns(schema, &numcolumns, error);
     while ((row_reader = esp_message_reader_next_row(reader,
error)) != NULL) {
         for (int i = 0; i < numcolumns; ++i) {
                rc = esp_schema_get_column_type(schema, i, &type,
error);
                switch(type){
            case ESP_DATATYPE_INTEGER:
                   rc = esp_row_reader_get_integer(row_reader, i,
&int_value, error);
                    break;
            case ESP_DATATYPE_LONG:
                    rc = esp_row_reader_get_long(row_reader, i,
&long_value, error);
                    break;
            case ESP_DATATYPE_DATE:
                    rc = esp_row_reader_get_date(row_reader, i,
```

```
&date_value, error);
                    break;
                }
        }
```

.Net 外部アダプタ

.Net SDK を使用して、カスタム .Net 外部アダプタを作成します。

参照：

- *カスタム・アダプタ* (3 ページ)
- *Java 外部アダプタ* (572 ページ)
- *C/C++ 外部アダプタ* (576 ページ)
- *カスタム外部アダプタの作成* (571 ページ)

サーバへの接続

サーバに接続する場合、クレデンシャルと .Net サーバ・オプションを設定します。

1. **NetEspError** コマンドを実行し、これらのタスクのエラー・メッセージ・ストアを作成します。
   ```
   NetEspError error = new NetEspError();
   ```
2. 新規 URI を設定します。
   ```
   NetEspUri uri = new NetEspUri();
   uri.set_uri("esp://cepsun64amd.sybase.com:19011", error);
   ```
3. クレデンシャルを作成します。
   ```
   NetEspCredentials creds = new
   NetEspCredentials(NetEspCredentials.NET_ESP_CREDENTIALS_T.NET_ESP
   _CREDENTIALS_SERVER_RSA);
   creds.set_user("pengg");
   creds.set_password("1234");
   creds.set_keyfile("..¥¥test_data¥¥keys¥¥client.pem");
   ```
4. オプションを設定します。
   ```
   NetEspServerOptions options = new NetEspServerOptions();
   options.set_mode(NetEspServerOptions.NET_ESP_ACCESS_MODE_T.NET_CA
   LLBACK_ACCESS);
   server = new NetEspServer(uri, creds, options);
   int rc = server.connect(error);
   ```

プロジェクトへの接続

サンプル .Net コードを使用してプロジェクトに接続します。

1. プロジェクトを取得します。
   ```
   NetEspProject project = server.get_project(“test”, “test”,
   error);
   ```
2. プロジェクトに接続します。

```
project.connect(error);
```

パブリッシュ
パブリッシャを作成し、ローを追加して、パブリッシュ・プロセスを完了します。

1. パブリッシャを作成します。
   ```
   NetEspPublisher publisher = project.create_publisher(null,
   error);
   ```
2. パブリッシャに接続します。
   ```
   Publisher.connect(error);
   ```
3. ストリームを取得します。
   ```
   NetEspStream stream = project.get_stream("WIN2", error);
   ```
4. メッセージ・ライタを取得します。
   ```
   NetEspMessageWriter writer = publisher.get_message_writer(stream,
   error);
   ```
5. ロー・ライタを取得、起動し、opcode を設定して 1 つのローを挿入します。
   ```
   NetEspRelativeRowWriter rowwriter =
   writer.get_relative_row_writer(error);
   rowwriter.start_row(error);
   rowwriter.set_opcode(1, error);
   ```
   (オプション) トランザクション・モードでのパブリッシュの場合、これらの引数を使用して複数のローを追加します。
   ```
   NetEspRelativeRowWriter rowwriter =
   writer.get_relative_row_writer(error);
   for(int i=0; i<100; i++){
       rowwriter.start_row(error);
   //add row columns' values
   rowwriter.end_row(error);
   }
   ```
6. データをパブリッシュします。
   ```
   rc = publisher.publish(writer, error);
   ```

サブスクライバへの接続
新規サブスクライバを作成、接続します。

1. サブスクライバを作成します。
   ```
   NetEspSubscriberOptions options = new NetEspSubscriberOptions();
   options.set_mode(NetEspSubscriberOptions.NET_ESP_ACCESS_MODE_T.NE
   T_CALLBACK_ACCESS);
   NetEspSubscriber subscriber = new NetEspSubscriber(options,
   error);
   ```
2. サブスクライバに接続します。
   ```
   Subscriber.connect(error);
   ```

コールバック・モードを使用したサブスクライブ
新しいデータのコールバックを実施します。

1. サブスクライバ・オプションを設定します。
```
NetEspSubscriberOptions options = new NetEspSubscriberOptions();
options.set_mode(NetEspSubscriberOptions.NET_ESP_ACCESS_MODE_T.NE
T_CALLBACK_ACCESS);
NetEspSubscriber subscriber = new NetEspSubscriber(options,
error);
```
2. コールバック・インスタンスを作成します。
```
NetEspSubscriber.SUBSCRIBER_EVENT_CALLBACK callbackInstance = new
NetEspSubscriber.SUBSCRIBER_EVENT_CALLBACK(subscriber_callback);
```
3. コールバック・レジストリを作成します。
```
subscriber.set_callback(NetEspSubscriber.NET_ESP_SUBSCRIBER_EVENT
.NET_ESP_SUBSCRIBER_EVENT_ALL, callbackInstance, null, error);
```
4. サブスクライバに接続します。
```
subscriber.connect(error);
```
5. ストリームにサブスクライブします。
```
subscriber.subscribe_stream(stream, error);
```
6. コールバックを実装します。
```
Public static void subscriber_callback(NetEspSubscriberEvent
event, ValueType
data) {
switch (evt.getType())
            {
                case (uint)
(NetEspSubscriber.NET_ESP_SUBSCRIBER_EVENT.NET_ESP_SUBSCRIBER_EVE
NT_CONNECTED):
                    Console.WriteLine("the callback happened:
connected!");
                    break;
(uint)
( NetEspSubscriber.NET_ESP_SUBSCRIBER_EVENT.NET_ESP_SUBSCRIBER_EV
ENT_DATA):
```
7. (オプション) **handleData** を使用して、サブスクライブされたデータを取得、使用するための別のメソッドを完成します。
```
NetEspRowReader row_reader = null;
while ((row_reader = evt.getMessageReader().next_row(error)) !=
null) {
               for (int i = 0; i < schema.get_numcolumns(); ++i) {
                    if ( row_reader.is_null(i) == 1) {
                        Console.Write("null, ");
                        continue;
                    }
                    switch
(NetEspStream.getType(schema.get_column_type((uint)i, error)))
                    {
```

```
                        case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_INTEGER:
                       ivalue = row_reader.get_integer(i, error);
                           Console.Write(ivalue + ", ");
                           break;
                        case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_LONG:
                          lvalue = row_reader.get_long(i, error);
                           Console.Write(lvalue + ", ");
                           break;
                        case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_FLOAT:
                         fvalue = row_reader.get_float(i, error);
                           Console.Write(fvalue + ", ");
                           break;
                        case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_STRING:
                         svalue = row_reader.get_string(i, error);
                           Console.Write(svalue);
                           break;
                        case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_DATE:
                           dvalue = row_reader.get_date(i, error);
                           Console.Write(dvalue + ", ");
                           break;
                        case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_TIMESTAMP:
                           tvalue = row_reader.get_timestamp(i,
error);
                           Console.Write(tvalue + ", ");
                           break;
                        case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_BOOLEAN:
                          boolvalue = row_reader.get_boolean(i,
error);
                           Console.Write(boolvalue + ", ");
                           break;
                        case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_BINARY:
                           uint buffersize = 256;
                           binvalue = row_reader.get_binary(i,
buffersize, error);

Console.Write(System.Text.Encoding.Default.GetString(binvalue) +
", ");
                           break;
                        case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_INTERVAL:
                         intervalue = row_reader.get_interval(i,
error);
                           Console.Write(intervalue + ", ");
                           break;
                        case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_MONEY01:
                           mon = row_reader.get_money(i, error);
                       Console.Write(mon.get_long(error) + ", ");
```

```
                            break;
                        case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_MONEY02:
                         lvalue = row_reader.get_money_as_long(i,
error);
                            Console.Write(lvalue + ", ");
                            break;
                        case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_MONEY03:
                            mon = row_reader.get_money(i, error);
                       Console.Write(mon.get_long(error) + ", ");
                            break;
                        case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_MONEY10:
                            mon = row_reader.get_money(i, error);
                       Console.Write(mon.get_long(error) + ", ");
                            break;
                        case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_MONEY15:
                            mon = row_reader.get_money(i, error);
                       Console.Write(mon.get_long(error) + ", ");
                            break;
                        case
NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_BIGDATETIME:
                            bdt2 = row_reader.get_bigdatetime(i,
error);
                       long usecs = bdt2.get_microseconds(error);
                            Console.Write(usecs + ", ");
                            break;
                    }
                }
            }
```

8. サブスクライバから切断します。

```
            rc = subscriber.disconnect(error);
}
```

第 4 章 スキーマ検出

スキーマ検出機能を使用して、外部スキーマを検出し、アダプタに接続されているデータソースからのデータのフォーマットに基づいて CCL スキーマを作成できます。

ストリームまたはウィンドウの各ローは同じ構造またはスキーマを持つ必要があります。これには、カラムの名前、カラムのデータ型、カラムの配列順序が含まれます。複数のストリームまたはウィンドウが同じスキーマを使用できますが、1 つのストリームまたはウィンドウは 1 つのスキーマしか持つことができません。

スキーマ検出を使用して、スキーマを検出し、アダプタに接続されているデータソースのデータのフォーマットに基づいてスキーマを自動的に作成できます。新しいスキーマを手動で作成する必要はありません。 たとえば、データベース・インプット・アダプタの場合、アダプタの接続先のデータベースから特定のテーブルに対応するスキーマを検出できます。

スキーマを検出するには、最初にアダプタ・プロパティを設定する必要があります。 スキーマ検出をサポートする各アダプタには、スキーマ検出を有効にするために設定する必要のある一意なプロパティがあります。

スキーマ検出をサポートするアダプタ

スキーマ検出をサポートするアダプタと、スキーマ検出を有効にするために使用されるプロパティ。

アダプタ	スキーマ検出のサポート	プロパティ
AtomReader インプット	不可	—
データベース・インプット	可	データベース・サービス アダプタがデータベース接続を取得するデータベース・サービスの名前。
データベース・アウトプット	可	データベース・サービス 使用するサービス・エントリの名前。

アダプタ	スキーマ検出のサポート	プロパティ
ファイル CSV インプット	可	ディレクトリ アダプタが読み込むデータ・ファイルの絶対パス。
ファイル CSV アウトプット	不可	—
FIX ファイル・インプット	不可	—
FIX ファイル・アウトプット	不可	—
XML ファイル・インプット	可	ディレクトリ アダプタが読み込むデータ・ファイルの絶対パス。
XML ファイル・アウトプット	不可	—
FIX インプット	不可	—
FIX アウトプット	不可	—
フレックス・アウトプット	不可	—
HTTP インプット	不可	—
JMS CSV インプット	可	• 区切り文字 – フィールド区切り文字 • 接続ファクトリ – 接続ファクトリのクラス名 • JNDI Context Factory – JNDI 接続を初期化するためのコンテキスト・ファクトリ • JNDI URL • 送信先タイプ • 送信先名
JMS CSV アウトプット	不可	—
JMS カスタム・インプット	不可	—
JMS カスタム・アウトプット	不可	—
JMS FIX インプット	不可	—

アダプタ	スキーマ検出のサポート	プロパティ
JMS FIX アウトプット	不可	—
JMS Object Array インプット	可	• 接続ファクトリ – 接続ファクトリのクラス名 • JNDI Context Factory – JNDI 接続を初期化するためのコンテキスト・ファクトリ • JNDI URL • 送信先タイプ • 送信先名
JMS Object Array アウトプット	不可	—
JMS XML インプット	可	• 接続ファクトリ – 接続ファクトリのクラス名 • JNDI Context Factory – JNDI 接続を初期化するためのコンテキスト・ファクトリ • JNDI URL • 送信先タイプ • 送信先名
JMS XML アウトプット	不可	—
Kdb インプット	可	• KDB サーバ • KDB ポート • KDB ユーザ • KDB パスワード
Kdb アウトプット	可	• KDB サーバ • KDB ポート • KDB ユーザ • KDB パスワード
ログファイル・インプット	不可	—
ランダム・タプル生成インプット	不可	—

アダプタ	スキーマ検出のサポート	プロパティ
ロイター・マーケットフィード・インプット	可	検索パス
ロイター・マーケットフィード・アウトプット	不可	—
ロイター OMM インプット	可	検索パス
ロイター OMM アウトプット	不可	—
RTView アウトプット	不可	—
SMTP アウトプット	不可	—
ソケット (クライアント側) CSV インプット	不可	—
ソケット (クライアント側) CSV アウトプット	不可	—
ソケット (クライアント側) XML インプット	不可	—
ソケット (クライアント側) XML アウトプット	不可	—
ソケット (サーバ側) XML インプット	不可	—
ソケット (サーバ側) XML アウトプット	不可	—
ソケット (サーバ側) CSV インプット	不可	—
ソケット (サーバ側) CSV アウトプット	不可	—
ソケット FIX インプット	不可	—
ソケット FIX アウトプット	不可	—
Sybase IQ アウトプット	不可	—
オープン・インプット／アウトプット	不可	—
Tibco Rendezvous インプット	不可	—

アダプタ	スキーマ検出のサポート	プロパティ
Tibco Rendezvous アウトプット	不可	—
NYSE インプット	可	ディレクトリ・パスの検索 アダプタ検出ディレクトリへの絶対パス。

参照：

- *内部アダプタ* (21 ページ)
- *外部アダプタ* (136 ページ)

第 5 章 保証された配信

保証された配信 (GD) は、ストリームからのデータがアダプタで処理されることを保証する配信メカニズムです。

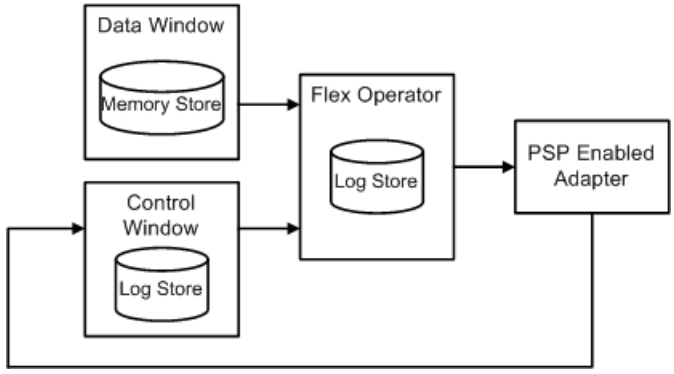
GD は、次の場合にデータの処理が継続されるようにします。

- サーバで障害が発生した場合。
- 送信先 (サードパーティのサーバ) で障害が発生した場合。
- 送信先 (サードパーティのサーバ) が一定期間応答しない場合。

永続サブスクライブ・パターン (PSP) は、出力アダプタで GD を実装するために使用されます。 インプット・アダプタは、PSP を使用するのではなく、ソースによって提供される機能を使用して GD をサポートします。 WebSphereMQ インプット・アダプタと WebSphereMQ アウトプット・アダプタ、すべての JMS インプット・アダプタと JMS アウトプット・アダプタ、TIBCO Rendezvous アダプタはいずれも GD をサポートします。 これらのアダプタには、それらに固有の PSP パラメータと GD パラメータがあります。 JMS CSV アウトプット・アダプタの 1 つと WebSphere アウトプット・アダプタで GD を有効にする例が、それぞれ `<ESP_HOME>/examples/ccl/JmsOutBoundAdapterWithGDSupport` と `<ESP_HOME>/examples/ccl/WsmqOutBoundAdapterWithGDSupport` にあります。

PSP は、ウィンドウ (入力、出力)、制御ウィンドウ、ログ・ストアを持つフレックス演算子の組み合わせによって機能します。 ウィンドウと制御ウィンドウは、フレックス演算子に接続します。 PSP が有効なウィンドウからのデータはフレックス演算子に入ります。フレックス演算子は、このデータからシーケンス番号と opcode を生成し、データの各ローの先頭に配置します。 フレックス演算子は、このデータを演算子に付加されているアダプタに送り、アダプタは、その情報を制御ウィンドウに渡します。 最後に、制御ウィンドウが、アダプタによって処理されたデータをフレックス演算子に通知し、フレックス演算子は、このデータをログ・ストアから削除します。

図 12 : PSP の概要



参照：

- *Tibco Rendezvous アダプタ* (528 ページ)
- *WebSphere MQ インプット・アダプタ* (129 ページ)
- *WebSphere MQ アウトプット・アダプタ* (132 ページ)
- *JMS CSV インプット・アダプタ* (57 ページ)
- *JMS CSV アウトプット・アダプタ* (60 ページ)
- *JMS カスタム・インプット・アダプタ* (64 ページ)
- *JMS カスタム・アウトプット・アダプタ* (68 ページ)
- *JMS FIX インプット・アダプタ* (73 ページ)
- *JMS FIX アウトプット・アダプタ* (75 ページ)
- *JMS Object Array インプット・アダプタ* (78 ページ)
- *JMS Object Array アウトプット・アダプタ* (82 ページ)
- *JMS XML インプット・アダプタ* (86 ページ)
- *JMS XML アウトプット・アダプタ* (89 ページ)
- *出力ストリーム・パラメータ* (538 ページ)

ログ・ウィンドウ

ログ・ウィンドウは、ログ・ストアに割り当てられるフレックス演算子で、保証された配信 (GD) メカニズムの核となります。

永続サブスクライブ・パターン (PSP) を使用する永続サブスクリプションの場合、出力アダプタを、対象となるストリームの代わりにログ・ウィンドウにアタッチします。 ログ・ウィンドウのストリーム定義は、対象となるストリームに属する

すべてのカラムと、2 つの追加カラムで構成されます。 これらの追加カラムは、gdKey (長整数) と gdOpcode (整数) です。

gdKey は、対象となるストリームに格納される opcode に関係なく、各イベントを一意に識別する、常に増加する値です。 これは、ログ・ウィンドウのキーとして機能します。 gdOpcode は、対象となるストリームで発生するイベントのオペレーション・コード (たとえば、INSERT、UPDATE、または DELETE) です。

ログ・ウィンドウには、2 つの入力があります。保証された形態でデータが配信される必要のあるストリーム (対象となるストリーム) とトランケート・ウィンドウです。 ログ・ウィンドウには、各入力に関連付けられたメソッドがあります。対象となるストリームに関連付けられたメソッドは次の機能を提供します。

1. イベントを受信するごとに、gdKey を 0 から始めて 1 ずつ増やす。 再起動時には、以前に出力したデータを自身で検査することによって得られる、最後に生成したシーケンス番号から始めます。
2. 受信イベントの opcode を決定する。
3. 上記 2 つの手順で決定された gdKey と gdOpcode を、対象となるストリームからの入力イベントのすべてのカラムと共に出力する。

トランケート・ウィンドウに関連付けられているメソッドは、ログ・ウィンドウ内でデータが無制限に増加するのを抑制します。 トランケート・ウィンドウでイベントが発生するごとに、このメソッドは、指定された gdKey 以下の gdKey を持ち、データ削除フラグが true に設定されている、ログ・ウィンドウ内のすべてのイベントを削除します。

トランケート・ウィンドウ

出力アダプタは、トランケート・ウィンドウを使用して、アダプタによって処理済みであり、安全に削除できるデータがどれであるかをログ・ウィンドウに通知します。

simpleKey (整数)、gdKey (長整数)、purge (ブール値) の 3 つのカラムがあります。simpleKey カラムは、現時点では単なる 0 または 1 のダミー値で、トランケート・ウィンドウのキーとしてのみ機能します。 gdKey カラムには、出力アダプタが正しく処理した gdKey の値があります。 ログ・ウィンドウはこのカラムを使用して、提供された値以下の gdKey を持つすべてのデータを削除します。 purge カラムについては、true の値は、ログ・ウィンドウ内のデータが削除される必要があることを示します。 このカラムは、出力アダプタによって更新されます。

このウィンドウをログ・ストアに割り当て、障害時にこのウィンドウのデータを確実にリカバリできるようにします。

索引

H

I

J

K

L

M

N

O

P

R

S

T

U

W

X

あ

い

う

お

か

き

く

こ

さ

し

す

せ

と

は

ふ

へ

ま