Informatica

Informatica® Application Integration
July 2021

ファイルコネクタガイド

Informatica Application Integration ファイルコネクタガイド
July 2021
2021 年 7 月

発行日: 2021-08-10

目次

序文

*アプリケーションの統合*用のファイルコネクタガイドは、ファイル接続の設定方法と使用方法について説明しています。このガイドは、環境内のファイルシステムを理解していること、およびアプリケーションの統合を使用して接続を作成する方法を理解していることを前提としています。

第 1 章

ファイルコネクタの紹介

この章では、以下の項目について説明します。

ファイルコネクタの概要

ファイルコネクタは、アプリケーションの統合とファイルシステムとの間の接続を実現します。これにより、新しいファイルがあるかどうかについてファイルシステムを監視したり、ファイルを移動したり、ファイルに対してコンテンツを読み書きしたり、処理済みファイルを扱うためのオプションを選択できます。ファイルコネクタは、より大規模な統合の一環としてログファイルを書き込む、.csv ファイルがあるかどうかについてファイルシステムを監視する、ファイルから区切りコンテンツを読み取る、一連のプロセスオブジェクトで生成された XML を使用するなどの用途で使用できます。

ファイルコネクタを使用してイベントソースおよびイベントターゲットを作成すると、Process Designer も一連のプロセスオブジェクトを作成します。これらにアクセスすることで、レコード数やファイルサイズなどのファイル情報を取得できます。

ファイルコネクタは、Informatica Secure Agent で実行されます。

このガイドは、Process Designer、プロセスオブジェクト、および操作するファイルシステムで使用可能なオプションに精通していることを前提としています。

ファイルコネクタの機能

ファイルコネクタを使用すると、テキスト、区切りファイルコンテンツ、XML、JSON などのファイルコンテンツを読み書きできます。設定可能なイベントソースまたはイベントターゲットのいずれかを使用することで、次を実行できます。

- ファイルフォルダおよびプロセスファイルのフォルダへの追加を監視する。
- 移動や削除などのファイル操作を実行する。
- 区切りファイルのコンテンツを解析して、プロセスオブジェクトを作成し、コンテンツを Process Designer で使用できるようにする。
- Process Designer から呼び出せるイベントサービス（イベントターゲット）を作成する。
- 一連のプロセスオブジェクトを区切り形式でシリアル化し、ファイルに保存する。
- プロセスオブジェクトを XML/JSON 形式にシリアル化する。

- ファイルに対してテキストコンテンツを読み書きする。
- バイナリまたは添付形式で、ファイルに対してバイナリコンテンツを読み書きする。
- JSON ファイルと XML ファイルを解析し、それらを一連のプロセスオブジェクトに変換する。

アプリケーション統合での区切りファイルの読み込みとファイルライタの設定に関するビデオを表示したり、サンプルアセットをダウンロードしたりするには、次のコミュニティ記事を参照してください。

https://network.informatica.com/videos/3293

ファイルコネクタの実装

ファイルコネクタを使用すると、単一のファイル接続を定義し、1 つ以上のイベントソースまたはイベントターゲットを含めることができます。これらのイベントソースやイベントターゲットは、その後 Process Designer で使用したり呼び出せるプロセスオブジェクトを生成します。

次のイベントソースタイプを使用できます。

- ファイルモニタ。新しいファイルがあるかどうかについてディレクトリを監視します。
- ファイルパーサー。新しいファイルがあるかどうかについてディレクトリを監視し、処理済みの各ファイルのファイルコンテンツを解析します。
- 区切りコンテンツパーサー。新しいファイルがあるかどうかについてディレクトリを監視し、処理済みの各ファイルの区切りファイルコンテンツを解析します。

次のイベントターゲットタイプを使用できます。

- ファイルライタ。ターゲットディレクトリにファイルを書き込みます。
- 区切りコンテンツライタ。ターゲットディレクトリに区切りファイルを書き込みます。

イベントソースとイベントターゲットはそれぞれ一意のプロパティを持つことができるため、ファイルコネクタは、新しいファイルがあるかどうかについて複数の異なるフォルダを監視し、他の複数のフォルダから区切りコンテンツを読み取り、別のフォルダに書き込むことができます。

関連する一連のタスクを 1 つの接続に統合したり、作業をいくつかの異なる接続に分けることもできます。

リポジトリファイル

ファイルコネクタは、ファイルベースのキャッシュを使用して、処理済みエントリのリストを保持します。

また、必要に応じて、「.aedata」と呼ばれる特別なフォルダに冪等リポジトリファイルも格納します。これらの「冪等」ファイルは、以前に処理されたファイルに関するデータを記録する安全なファイルです。コネクタは、次の場合にのみ、これらのファイルを作成します。

1. 接続に対して［ファイル操作なし］オプションを有効にした場合。
2. 処理済みファイルを削除したり、別の場所に移動しない場合。

注: ファイルを処理後に移動する場合は、それらが処理中に監視対象のファイルパスにないようにします。ファイルはピリオドで始まる名前のフォルダ（「.done」や「.error」など）に移動することもできます。これによりコネクタは、以降のフォルダスキャンで、移動済みファイルをスキップします。

サーバーを再起動したり、接続を再パブリッシュすると、リポジトリリストのメモリ内コピーは使用できなくなり、コネクタはファイルからリストをロードします。

リポジトリファイル名の形式は次のとおりです。

```
.aedata/tenant_connectionName_eventSourceName.Aerepo
```

このファイルは、ファイルに対する変更を検出するため、相対パス、ファイルサイズ、および変更タイムスタンプを格納します。

コネクタは、リポジトリファイルをロードするたびにリポジトリのクリーンを実行し、リポジトリサイズを縮小して、存在しなくなったファイルのエントリを削除します。以前に削除されたファイルを再び処理するには、それらを監視対象フォルダにコピーします。

第 2 章

ファイル接続

基本的な接続プロパティ

次の表に、接続の作成ページの **[プロパティ]** タブで設定可能な基本プロパティを示します。

プロパティ	説明
名前	必須。Process Designer での識別に使用される、ファイル接続の一意の名前。名前はアルファベットで始まり、アルファベット、数値、ハイフン（-）のみを含めることができます。
場所	オプション。接続を保存するプロジェクトまたはフォルダの場所。**[参照]** をクリックして場所を選択します。 [Explore（参照）] ページが現在アクティブになっていて、プロジェクトまたはフォルダが選択されている場合、接続のデフォルトの場所はその選択されているプロジェクトまたはフォルダです。そうでない場合、デフォルトの場所は直近で保存されたアセットの場所です。
説明	オプション。接続の説明。
タイプ	必須。コネクタまたはサービスコネクタに使用する接続のタイプ。 **[ファイル]** を選択します。
曜日の指定	必須。接続を実行する必要がある Secure Agent グループまたは Secure Agent マシンの名前。
接続テスト	ファイルコネクタではサポートされていません。
OData 対応	ファイルコネクタではサポートされていません。

基本プロパティを設定した後、次のプロパティも定義する必要があります。

- ファイル接続タイプに適用できるプロパティ
- ファイル接続のイベントソースプロパティとイベントターゲットプロパティ

[メタデータ] タブには、ファイル接続のパブリッシュ時に生成されたプロセスオブジェクトが表示されます。

第 3 章

ファイルイベントソース

この章では、以下の項目について説明します。

基本的なイベントソースのプロパティ

接続に 1 つ以上のイベントソースを追加できます。イベントソースは、指定した場所の新しいファイルまたはメッセージをリスンまたは監視する開始イベントとして機能します。接続のイベントソースを定義した後は、クラウドサーバーではなく Secure Agent でのみ接続をパブリッシュできます。その後、プロセス内のイベントソースにアクセスし、イベントソースダウンストリームが生成したプロセスオブジェクトを消費する目的で、プロセスを Secure Agent 上でのみデプロイできます。

接続のイベントソースを作成するには、**[イベントソース]** タブで **[イベントソースの追加]** をクリックし、使用可能なリストからイベントソースタイプを選択します。

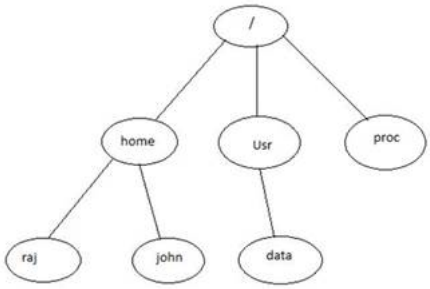
以下の表に、すべてのイベントソースタイプに使用できるイベントソースプロパティを示します。

プロパティ	説明
名前	必須。イベントソースの一意の名前。
説明	オプション。イベントソースの説明。
有効	このイベントソースをパブリッシュ後即時に使用できるようにするには、**[はい]** を選択します。 このイベントソースを使用準備ができるまで無効にするには、**[いいえ]** を選択します。 デフォルトは **[はい]** です。

ファイルの場所のプロパティ

次の表に、ファイルの場所のプロパティを示します。

注: ファイルを含める、または除外するフィルタを定義するために正規表現を使用する場合は、https://regex101.com/を参照し、ここにあるようなツールを使用して式の構文が正しいことを確認したり、Java 形式の正規表現を学ぶことができます。

プロパティ	説明
切断（FTP 接続および SFTP 接続のみ）	イベントソースがディレクトリをポーリングするたびにリモート FTP サーバーから切断するかどうを決定します。イベントソースがリモートディレクトリを確認する頻度が少ない場合は、[はい] を選択して、ポーリングイベントが発生するたびに接続を切断して新しい接続を確立します。 デフォルト: いいえ
ディレクトリ	必須。監視するディレクトリの相対パスを指定します。 例: data/docs 疑問符（?）は、このフィールドでは使用できない文字です。 FTP または SFTP 接続を使用する場合、必ず FTP または SFTP サーバー上の FTP または SFTP ホームディレクトリに対する相対パスを指定します。GUI クライアントを使用して FTP または SFTP エンドポイントにアクセスすると、ホームディレクトリを見つけることができます。ホームディレクトリは、FTP または SFTP エンドポイントにアクセスしたときに接続される最初のディレクトリです。 例えば次のディレクトリ構造の場合、raj がホームディレクトリです。  data ディレクトリを指定する場合は、次のように **[ディレクトリ]** フィールドに相対パスを指定する必要があります。 ../../usr/data FTP 接続は、raj ディレクトリからルートディレクトリまで 2 つディレクトリを戻ってから、usr ディレクトリ、続いて data ディレクトリに移動します。
再帰的	イベントソースが、指定したディレクトリ内にあるすべてのサブディレクトリをファイルの検索対象にするかどうかを決定します。 このオプションは慎重に使用する必要があります。例えば、処理済みのファイルをサブディレクトリに移動すると、これらのファイルが再び監視されることになります。その結果、ネストされた一連のディレクトリが繰り返し作成されるなどの予期しない結果を生む場合があります。 **注:** コネクタは、ピリオド（「.」）で始まるディレクトリ名を無視します（例: 「.done」）。この文字は、特定のディレクトリ内にあるすべてのファイルをスキップする場合のプレフィックスとして使用できます。ベストプラクティスは、処理済みファイルをサブディレクトリに格納するためにこの方法を使用し、再帰の問題を回避することです。

プロパティ	説明
ファイルを含む	オプション。処理が必要なファイルを選択する正規表現を入力します。Java 形式の正規表現を使用します。 例えば、拡張子が.txt のファイルのみを選択するには、次のような正規表現になります。 。*\.txt スラッシュの代わりにバックスラッシュを使用しています。
ファイルを除外する	オプション。除外する必要のあるファイルを選択する正規表現を入力します。Java 形式の正規表現を使用します。

ファイル操作とポーリングのプロパティ

次の表に、各イベントソースのファイル操作とポーリングのプロパティを示します。

プロパティ	説明
ファイル操作なし	読み取り専用データを処理するために、ファイルの移動または削除を禁止するかどうかを決定します。 デフォルトは **［いいえ］** です。
処理済みファイルの削除	各ファイルを、正常に処理した後で削除するかどうかを決定します。 **注**: このオプションを［移動先］と一緒に使用しないでください。 デフォルトは **［いいえ］** です。
移動前	オプション。新しいファイルの名前と場所を処理前に動的に設定する正規表現を入力します。
移動先	オプション。ファイルの名前と、ファイルが正常に処理された後に移動する場所を動的に設定する正規表現を入力します。 ［再帰的］プロパティを有効にした場合は、移動済みファイルを繰り返し処理することがないように、このオプションを慎重に使用します。 デフォルトは.done です。 **注**: このオプションを［処理済みファイルの削除］と一緒に使用しないでください。
失敗した場合に移動	オプション。［移動先］操作が失敗した場合に、別のターゲットディレクトリを動的に設定するために使用する正規表現を入力します。 デフォルトは.error です。 **注**: このオプションを［処理済みファイルの削除］と一緒に使用しないでください。
初期遅延	ポーリングを開始するまでの秒数。 デフォルトは 1 秒です。
ポーリング間隔	新しいファイルが到着したかどうかを確認するために、次のポーリングまで待機する秒数。 デフォルトは 1 秒です。

プロパティ	説明
ポーリングあたりの最大メッセージ数	場所をポーリングするたびに取得するオブジェクトの最大数。上限を設定しない場合は 0 と入力します。 デフォルトは 0 です。
その他の属性	オプション。このイベントソースタイプで使用できる他のパラメータの一覧を指定できます。属性はコネクタによってエンコードされるため、URI エンコードされた形式で属性を入力する必要はありません。 **注**: コネクタで公開済みの特定の属性（ポーリング間隔など）を追加する場合、Process Designer はそれらの属性を無視します。

ファイル読み取りロックのプロパティ

次の表に、ファイル読み取りロックのプロパティを示します。

プロパティ	説明
読み取りロック	必須。 ファイル読み取りロックモードを選択して、イベントソースがファイルにアクセスしているときに、使用中のファイルを同時に処理しないようにします。イベントソースは、ファイルロックが許可されるまで待機します。オプション: - changed: ファイルの長さと変更タイムスタンプを使用して、ファイルが現在コピー中かどうかを検出します。このオプションを使用すると、変更の検出に必要な遅延が原因で処理速度が低下します。 - rename: ファイルの名前をテストとして変更し、排他的な読み取りロックが可能かどうかを決定します。デフォルト。 - none: 読み取りロックを使用しません。 一部の接続タイプ（ファイルコネクタなど）では、これらの読み取りロックのオプションも使用可能です。 - markerFile: マーカーファイルを作成し、そのファイルのロックを保持します。 - fileLock: ファイルの読み取りロックを取得します。 マウントまたは共有を使用してリモートファイルシステムにアクセスする場合、ファイルシステムが分散ファイルロックをサポートしていない限りこの方法はお勧めしません。
読み取りロックタイムアウト	読み取りロックを取得できない場合に、現在のファイルをスキップするまで待機する秒数を入力します。 次のポーリング中に、イベントソースはスキップされたファイルの処理を再度試みます。0 に設定すると、イベントソースは必要とされる期間待機します。 デフォルト: 10 秒。
読み取りロックチェック間隔	ファイルの読み取りロックを次に取得するまで待機する秒数を入力します。 デフォルト: 1 秒。 **注**: イベントソースが各ファイルに対するファイルロックの試みを完了できるよう、［読み取りロックタイムアウト］はこのプロパティ値の 3 倍以上に設定します。

プロパティ	説明
読み取りロック最小長	イベントソースが処理する最小ファイルサイズをバイト単位で入力します。 **注**: このプロパティは、**［読み取りロック］**が［changed］に設定されている場合にのみ使用します。 デフォルト: 1 バイト
読み取りロックログレベル	必須。 読み取りロックを取得できないときに使用するログレベル（オフ、情報、警告、またはエラー）を選択します。 デフォルト: 警告

ファイル解析およびコンテンツタイプのプロパティ

使用している接続タイプに対応する特定のイベントソースタイプに基づいて、イベントソースを設定できます。

イベントソースのプロパティでは、コンテンツを解析するときにそれをどのように処理するかを決定します。イベントソースのパブリッシュ後、接続は新しいファイルを待機し、その到着時にコンテンツを解析し、次のプロパティに基づいてコンテンツを表現します。

- ファイル解析
- コンテンツタイプ

次の表に、ファイル解析のプロパティを示します。

プロパティ	適用対象	説明
文字セット	すべてのイベントソースタイプ	必須。ファイルのエンコーディングを決定します。ノードが指定されている場合、コネクタは UTF-8 を使用します。
区切り文字	区切りコンテンツパーサー	必須。区切りファイルの区切り文字。 区切り文字としてスペースまたはタブ文字を指定するには、エスケープ文字（「\s」、「\t」）を使用します。
テキスト修飾子	区切りコンテンツパーサー	必須。区切りファイルのテキスト修飾子。 修飾子としてスペースまたはタブ文字を指定するには、エスケープ文字（「\s」、「\t」）を使用します。

プロパティ	適用対象	説明
最初のレコードを無視	区切りコンテンツパーサー	コネクタが、区切りファイルの最初の行をデータ行として処理するかどうかを決定します。以下のオプションから選択できます。 - **はい**: コネクタは、区切りファイルの最初の行をデータ行として処理しません。 - **いいえ**: コネクタは、区切りファイルの最初の行をデータ行として処理します。**［いいえ］**を選択した場合は、［カラム記述子］属性を使用してカスタムヘッダーを指定する必要があります。 デフォルト値は **［はい］**です。
行を分割	区切りコンテンツパーサー 固定幅コンテンツパーサー	各行を個別に処理し、各行をプロパティオブジェクトに変換し、それぞれについて別個のイベントを生成するかどうかを決定します。

カラム記述子	区切りコンテンツパーサー
	固定幅コンテンツパーサー
		処理する固定幅カラムヘッダーを定義します。値は、名前と括弧で囲んだサイズをカンマ区切りのリストとして指定します。
		区切りコンテンツパーサーの例:
		`User Name, Password, Email`
		固定幅コンテンツパーサーの例:
		`User Name(40), Password(8), Email(14)`
		カスタムオブジェクトを使用する場合、これらのカラムヘッダーは、別の一連のヘッダーを指定しない限り、プロセスオブジェクトのヘッダー名も決定します。(以下を参照)。

カラムの不整合を無視	区切りコンテンツパーサー 固定幅コンテンツパーサー	区切りコンテンツパーサーの場合、次を決定します。 - データ行の余分なカラムを無視するかどうか。 - カラムが欠落している場合に、空の文字列値があるカラムを追加するかどうか。 有効にしない場合、イベントソースは、余分なカラムまたは欠落したカラムを見つけると例外をスローします。 固定幅コンテンツパーサーの場合、次を決定します。 - 行の余分な文字を無視するかどうか。 - 定義済みのカラムに欠落した値がある場合に、空のスペースがある文字を追加するかどうか。 デフォルト: いいえ。

次の表に、すべてのコンテンツタイプのプロパティを示します。

プロパティ	適用対象	説明
コンテンツ形式	ファイルパーサー ファイルモニタ FTP モニタ	必須。処理するコンテンツの形式。 - **無視**。コンテンツを処理しません。 - **プレーンテキスト**。コンテンツは文字列です。 - **バイナリ**。コンテンツは Base64 でエンコードした文字列にする必要があります。 - XML **および** JSON。コンテンツは、解析し、オブジェクトまたはプロセスオブジェクトの一覧に変換する必要があります。 - **添付**。コンテンツは添付です。
コンテンツの簡素化	ファイルパーサー ファイルモニタ FTP モニタ	有効にすると、イベントソースは、有効なプロセスオブジェクト構造と一致しない XML/JSON コンテンツを解析し、プロセスオブジェクトの形式と一致するようにコンテンツ構造の変更を試みます。 デフォルト: いいえ
単一オブジェクトモード	ファイルパーサー ファイルモニタ FTP モニタ	有効にすると、XML コンテンツは単一オブジェクトに変換されます。 デフォルト: いいえ

プロパティ	適用対象	説明
組み込みプロセスオブジェクトの使用	区切りコンテンツパーサー 固定幅コンテンツパーサー	異なるフィールドセットを使用するファイルを扱っている場合、またはファイルヘッダーが事前に分からない場合は、**[はい]** をクリックします。ファイルコネクタは、ファイルコンテンツに基づいて、レコードをプロセスオブジェクトのセットとして表します。 カスタムオブジェクトにフィールド名のリストを指定する場合は、**[いいえ]** を選択します。これは、類似ファイルを扱っており、各ファイルにフィールドセット（ヘッダー）があると確信できる場合に適しています。イベントソースは、区切りコンテンツファイルの各レコードについて単純なプロセスオブジェクトを生成します。デフォルトは［いいえ］です。 プロセスオブジェクトの操作の詳細については、「イベントプロセスオブジェクトの概要」 (ページ 22)を参照してください。
カスタムオブジェクト	区切りコンテンツパーサー 固定幅コンテンツパーサー	［組み込みプロセスオブジェクトの使用］を選択しない場合は、ファイルコンテンツを表すプロセスオブジェクトヘッダー名のカンマ区切りのリストを入力します。この方法により、必要なデータのみが抽出されます。生成されたオブジェクトの方が、より簡単に扱え、プロセスで処理するコードも少なくて済みます。 名前は、ファイル内のヘッダー名またはカラム記述子で定義されたヘッダーと完全に一致する必要があります。以下に例を示します。 `Name,Street,City,State,Postal Code,Country,Phone` **注**:「index」をフィールド名として使用することはできません。これは行インデックス情報用に予約されています。 ファイルの解析時にここで入力したヘッダーが存在しない場合、関連フィールドは生成されたオブジェクト内で空になります。 カスタムオブジェクトは *NCName* 形式で表されます。その際、有効なプロセスオブジェクトフィールド名になるように、禁止文字はヘッダー名から削除されます。

カスタムオブジェクトのエスケープ文字

カラムヘッダーでカンマ文字を指定する必要がある場合は、エスケープ文字としてバックスラッシュ（\）を追加し、ファイルが正しく解析されるようにします。例えば、ファイルに次が含まれているとします。

```
First, Name, Last Name, Address, local, Phone
```

この場合、バックスラッシュにより、コンテンツの解析時にカンマが区切り文字として使用されません。

```
First\, Name, Last Name, Address\, local, Phone
```

固定幅ファイル

固定幅パーサーでは、カラムヘッダーがない、または区切り文字がない固定幅ファイルを処理できます。イベントソースプロパティで、固定幅コンテンツを表すプロセスオブジェクトを生成できるカラム記述子を定義できます。

例えば、カラム記述子プロパティ内で、データファイルの構造を定義し、カラム名やカラム長のリスト（括弧で囲む）を入力できます。

```
 ID(3), User Name(100), Password(20), Nick Name(25), Email(20)
```

これにより、固定幅パーサーが固定幅ファイルを読み取り、これらの値に基づいてレコードを構成するプロセスオブジェクトを生成できます。これは、カラム幅があることを加えて、区切りコンテンツパーサーと類似しています。これらの値によって、カスタムのプロセスオブジェクトを使用する場合のフィールド名セットのデフォルトも定義されます。

組み込みおよびカスタムのプロセスオブジェクトの両方が、イベントソースタイプに使用できます。

コネクタがデータの整合性を検証するため、指定したカラム記述子を超える、またはそれに足りない行が固定幅ファイル内の行にある場合、例外エラーが発生することがあります。行の余分な文字を無視し、文字数が指定に足りない行の末尾に空白を追加するには、［カラムの不整合を無視］プロパティを有効にできます。

第 4 章

ファイルイベントターゲット

この章では、以下の項目について説明します。

ファイルコネクタのイベントターゲット

ファイルコネクタには次のイベントターゲットタイプが含まれています。

- ファイルライタ。指定したディレクトリにあるファイルに、プレーンテキスト、バイナリ、添付、JSON、または XML 形式でデータを書き込みます。
- 区切りコンテンツライタ。プロセスオブジェクトを区切りコンテンツ形式にシリアル化し、指定したディレクトリにあるファイルに結果を書き込みます。

基本的なイベントターゲットのプロパティ

定義する接続ごとに、ファイルまたはメッセージを書き込むための操作、またはイベントターゲットがプロセスから呼び出された場合の操作を指定する 1 つ以上のイベントターゲットを含めることができます。例えば、プロセスオブジェクトから読み取るイベントターゲットを定義し、カンマ区切りファイルに書き込めます。

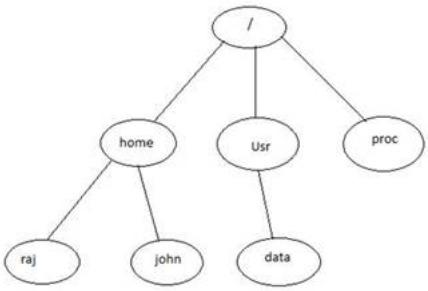
接続のイベントターゲットプロパティを設定するには、**[イベントターゲット]** タブで **[イベントターゲットの追加]** をクリックし、使用可能なリストからイベントターゲットタイプを選択します。

以下の表に、基本的なプロパティを示します。

プロパティ	説明
名前	必須。イベントターゲットの一意の名前。
説明	オプション。イベントターゲットの説明。

イベントターゲットファイルのプロパティ

次の表に、ファイルの場所のプロパティを示します。

プロパティ	説明
ディレクトリ	必須。ファイルを格納するディレクトリの相対パスを指定します。 例: data/docs 疑問符（?）は、このフィールドでは使用できない文字です。 FTP または SFTP 接続を使用する場合、必ず FTP または SFTP サーバー上の FTP または SFTP ホームディレクトリに対する相対パスを指定します。GUI クライアントを使用して FTP または SFTP エンドポイントにアクセスすると、ホームディレクトリを見つけることができます。ホームディレクトリは、FTP または SFTP エンドポイントにアクセスしたときに接続される最初のディレクトリです。 例えば次のディレクトリ構造の場合、raj がホームディレクトリです。  data ディレクトリを指定する場合は、次のように **[ディレクトリ]** フィールドに相対パスを指定する必要があります。 ../../usr/data FTP 接続は、raj ディレクトリからルートディレクトリまで 2 つディレクトリを戻ってから、usr ディレクトリ、続いて data ディレクトリに移動します。 **注**: プロセスが作成できるのはネストされたフォルダのみです。
ファイルの存在	必須。 同じ名前のファイルがすでに存在している場合の処理を決定します。 - **上書き**（デフォルト）: 既存のファイルを置き換えます。ターゲットが生成されたときに既存のファイルをどのように処理するかを決定するには、[削除の強制] オプションを選択します。 - **付加**: 既存のファイルにコンテンツを追加します。一時的なプレフィックスや一時的なファイル名は指定しないでください。 - **失敗**: 競合ファイルをスキップして、ファイル名がすでに存在することを示す例外をスローします。 - **無視**: 競合ファイルをスキップして、問題を無視します（例外のスローなし）。 - **移動**: ターゲットファイルに書き込む前に既存のファイルを移動します。既存のファイルの移動先フォルダを指定するには、[既存の移動] を使用します。 - **名前の変更**: ファイル名が存在するかどうかを確認せずに、ファイルの名前を一時的な名前から実際の名前に変更します。[一時的なファイル名] も指定している場合にのみ使用します。一部のファイルシステムまたは FTP サーバーでは、この方が早い場合があります。

プロパティ	説明
既存の移動	［ファイルの存在］の［移動］オプションと一緒に使用すると、ターゲットに書き込む際に既存のファイルの場所を指定できます。 このフィールドに「backup」と入力するだけで、既存のフィールドをバックアップフォルダに移動できます。バックアップファイルの名前を移動時に変更し、以降の操作でバックアップファイルが新しいバージョンに置き換わらないようにするには、ファイル名を決定するための式を入力します。以下に例を示します。 `backup\${file:name}_${file:modified}` この式は、既存のファイルが移動されるたびに新しいファイルをバックアップフォルダに作成し、ファイル名にタイムスタンプを追加します。 Apache Camel File Expression 言語の使用方法については、次を参照してください。 http://camel.apache.org/file-language.html
削除の強制	一時ファイルに書き込む前にターゲットファイルを削除するには、**［はい］**を選択します。この場合、［ファイルの存在］オプションに［上書き］を選択し、［一時的なファイル名］を指定する必要もあります（以下を参照）。 待機して、一時ファイルに書き込んで出力ファイルの名前に変更する準備が整った場合にのみターゲットファイルを削除するには、**［いいえ］**を選択します。これは、書き込み操作が完了するまでの間、既存のファイルを使えるようにする場合などに使用できます。

次の表に、ファイル書き込みのプロパティを示します。

プロパティ	説明
一時的なプレフィックス	ファイルに一時的な名前を付ける場合に、ファイル名のプレフィックスを入力します。書き込み操作が完了したら元の名前に戻します。 **注**: ［ファイルの存在］オプションに［付加］を選択した場合は無視されます。
一時的なファイル名	プレフィックスの代わりに、一時ファイル名を決定するための式を入力します。Apache Camel File Expression 言語を活用できます。 **注**: ［ファイルの存在］オプションに［付加］を選択した場合は無視されます。
書き込みの強制	ファイルシステムがすべてのデータをターゲットファイルに書き込むようにし、システム障害が発生した場合はすべてのデータが維持されるようにするには、**［はい］**を選択します。 ログデータを扱っており、ファイルシステム障害が発生した場合のデータフラグメントの損失も気にしない場合は、**［いいえ］**を選択します。これにより、パフォーマンスがやや向上します。
書き込みバッファサイズ	書き込みバッファのサイズ（バイト単位）を決定します。 デフォルト: 128kb。
文字セット	必須。ファイルのエンコーディングを決定します。 指定しない場合、イベントターゲットは UTF-8 のファイルエンコーディングを使用します。

区切りコンテンツライタのプロパティ

区切りコンテンツライタを使用する場合、既存のプロセスオブジェクトを区切りファイル形式にシリアル化できます。区切りイベントソースが作成したプロセスオブジェクトをシリアル化することもできます。

次の表に、区切りコンテンツライタのプロパティを示します。

プロパティ	説明
区切り文字	必須。このイベントターゲットが生成したファイルで使用するための区切り文字。 **注**: スペースまたはタブを使用するには、エスケープ文字（「\s」または「\t」）として入力します。 デフォルトは［,］です。
テキスト修飾子	必須。このイベントターゲットが作成したファイルで使用するテキスト修飾子。 **注**: スペースまたはタブを使用するには、エスケープ文字（「\s」または「\t」）として入力します。 デフォルトは［"］です。
ヘッダーのスキップ	必須。生成したコンテンツにヘッダー名があるかどうかを確認します。ヘッダー名なしで区切りコンテンツを保存する場合は**［はい］**を選択します。 デフォルトは**［いいえ］**です。
行の末尾のスタイル	必須。生成されたコンテンツでの行の末尾のスタイルを決定します（Windows スタイルは\r\n、Unix スタイルは\n）。 デフォルトは［Windows］です。

第 5 章

ファイルコネクタを使用したプロセスオブジェクト

この章では、以下の項目について説明します。

イベントプロセスオブジェクトの概要

プロセスオブジェクトとは一連の構造化されたデータで、これにより、サービスに送信されたデータやサービスから返されたデータを処理できます。

プロセスオブジェクトは、各接続に定義されたイベントソースとイベントターゲットに基づいて Process Designer が生成します。これらのプロセスオブジェクトは、接続にアクセスするプロセスで使用できます。

イベントソースプロセスオブジェクトは、キューまたはファイルシステムを監視し、ファイル操作をトリガする開始イベントとして動作します。イベントソースをパブリッシュすると、コネクタが、指定したディレクトリの監視を開始します。コネクタは、新しいファイルを検出すると、コンテンツを解析してそれをプロセスオブジェクトに変換し、これらのイベントをリスニングするプロセスにプロセスオブジェクトを送信します。

注: デフォルトでは、ディレクトリ監視を開始したパブリッシュ済みプロセスがなくても、接続のパブリッシュ時にイベントの生成が開始されます。監視の開始時間は、［初期遅延］設定を使用して延期できます。

イベントターゲットプロセスオブジェクトは、特定の形式でコンテンツを生成するために使用できるイベントサービスとして動作します。イベントターゲットは、パブリッシュした後で、Process Designer 内のプロセスから呼び出せます。イベントターゲットはイベントサービスと呼ばれることもあります。

ファイルコネクタのプロセスオブジェクトの概要

次のプロセスオブジェクトがファイルコネクタで使用されます。

- DelimitedContent。組み込みプロセスオブジェクトを使用する場合に、解析済みファイルから生成されます。この場合、フィールド、ヘッダー、レコードのプロセスオブジェクトも作成されます。
- <EventSource>Content（レコードおよび関連情報の集合）および<EventSource>Record（単一行）。カスタムオブジェクトを使用する場合に、解析済みファイルから生成されます。
- FileInformation。イベントソースが読み取る各ファイルについて生成されるファイルメタデータ。
- FileWriteTask。ファイルライタ（イベントターゲット）サービスへの要求を表します。
- SerializeToDelimitedContentTask。区切りコンテンツライタ（イベントターゲット）サービスへの要求を表します。
- SerializeToDelimitedContentResult。区切りコンテンツライタサービスからの応答として返されます。

ファイル接続を使用するプロセスを定義する場合に、これらのプロセスオブジェクトを使用できます。

また、これらのプロセスオブジェクトは、ファイルコネクタが扱うファイルのコンテンツを表します。

- PlainFileContent。プレーンテキストまたはバイナリファイルのコンテンツ。
- ParsedFileContent。プロセスオブジェクトリスト（または単一オブジェクト）で、XML または JSON で表わされるファイルのコンテンツ。
- AttachmentFileContent。添付として示されるファイルのコンテンツ。

組み込みプロセスオブジェクトの出力

注: 区切りコンテンツパーサーおよび固定幅コンテンツパーサーにのみ該当

区切りコンテンツと固定長ファイルを含む組み込みプロセスオブジェクトを選択すると、出力が DelimitedContent プロセスオブジェクトに示されます。例えば、いくつかのレコードを持つ単純な.csv ファイルを処理する場合に、ファイルが次のようになっていることがあります。

```
User Name,Age,Born
Bob Smith,22,1987
Bill White,45,1999
```

この.csv ファイルに基づき、ファイルのコンテンツは、ヘッダー名と各フィールドの値で表されます。プロセスオブジェクトは、ソースファイル、ヘッダーオブジェクトの一覧、レコードの一覧、行の合計数に関するファイル情報を格納します。

注: 行分割モードで作業している場合、ファイルは別々の行に分けられ、各行には、Process Designer が生成した DelimitedContent プロセスオブジェクトがヘッダーと 1 つのレコードで示されます。

```
<DelimitedContent>
     <fileInfo>
        <!-- for FTP/SFTP only -->
        <host>127.0.0.1</host>

        <lastModified>2015-04-29T14:37:35.448Z</lastModified>
        <dir>DataFiles</dir>
        <name>users</name>
        <path>/DataFiles/users.csv</path>
        <fullName>users.csv</fullName>
        <ext>csv</ext>
        <size>78</size>
     </fileInfo>
     <header>
```

```
        <name>User Name</name>
        <fieldIndex>1</fieldIndex>
     </header>
     <header>
        <name>Age</name>
        <fieldIndex>2</fieldIndex>
     </header>
     <header>
        <name>Born</name>
        <fieldIndex>3</fieldIndex>
     </header>
     <record>
        <field><value>Bob Smith</value></field>
        <field><value>22</value></field>
        <field><value>1987</value></field>
        <index>1</index>
     </record>
     <record>
        <field><value>Bill White</value></field>
        <field><value>45</value></field>
        <field><value>1999</value></field>
        <index>2</index>
     </record>
     <totalRowsCount>3</totalRowsCount>
</DelimitedContent>
```

カスタムオブジェクトフィールドの出力

注: 区切りコンテンツパーサーおよび固定幅コンテンツパーサーにのみ該当

カスタムオブジェクトフィールドを指定する場合:

- 解析されたファイルに見つからないヘッダーは、出力では空の要素として示されます。フィールドをスキップするには、［イベントソース］プロパティのカスタムオブジェクトフィールド一覧から対象のフィールドを除外します。
- 出力内のファイル情報はそのままです。
- プロセスオブジェクト名は、<sourceName>Content のようになり、レコードオブジェクトや他の詳細情報の集合を表します。
- いくつかの区切りコンテンツや固定長イベントソースがある場合、それぞれは独自のカスタムコンテンツやレコードオブジェクトを使用します。レコードオブジェクトは、<sourceName>Record 形式を使用します。
- フィールド名は NCName 形式に変換されます。その際、有効なプロセスオブジェクトフィールド名になるように、禁止文字はヘッダー名から削除されます。

例えば、次の.csv ファイルを処理する場合は、「Age, Born, User Name, Salary」というヘッダー名のカスタムオブジェクトフィールドを指定します:

```
Age,Born,User Name
22,1987,Bob Smith
45,1999,Bill White
```

この場合、結果は次に示す例のようになります。<Salary>要素が空なのは、これがカスタムオブジェクトフィールドとして指定されたものの、ソースファイルに「Salary」ヘッダーがないためです。

```
<MyUserFileContent>
     <fileInfo>
          <!-- for FTP/SFTP only -->
          <host>127.0.0.1</host>

          <lastModified> 2015-04-29T14:37:35.448Z </lastModified>
```

```
        <dir>DataFiles</dir>
        <name> users </name>
        <path>/DataFiles/users.csv </path>
        <fullName> users.csv </fullName>
        <ext> csv </ext>
        <size> 78 </size>
    </fileInfo>
    <record>
        <Age> 22 </Age>
        <Born> 1987 </Born>
        <Salary/>
        <User_Name> Bob Smith </User_Name>
    </record>
    <record>
        <Age> 45 </Age>
        <Born> 1999 </Born>
        <Salary/>
        <User_Name> Bill White </User_Name>
    </record>
    <totalRowsCount>3</totalRowsCount>
</MyUserFileContent>
```

ファイル情報プロセスオブジェクト

各イベントソースの FileInformation プロセスオブジェクトは、コンテンツではなくファイルの情報を格納します。

注: ファイルパス文字列は、パスの区切り文字として常にスラッシュを使用します。タイムスタンプは常に UTC 形式です。

以下に例を示します。

```
<FileInformation>
    <!-- for FTP/SFTP only -->
    <host>127.0.0.1</host>

    <!-- dateTime : file modified time -->
    <lastModified> 2015-04-29T14:37:35.448Z</lastModified>
    <!-- string: full path and name of resource. File separator is normalized to forward slash. -->
    <path>path/documents/users.csv</path>
    <!-- string: path to parent directory -->
    <dir>path/documents/</dir>
    <!-- string: file name with extension -->
    <fullName> users.csv</fullName>
    <!-- string: separate name only and extension only -->
    <name> users</name>
    <ext> csv</ext>
    <!-- double: file size in bytes -->
    <size> 78</size>
</FileInformation>
```

イベントターゲットプロセスオブジェクト

イベントターゲットの定義により、プロセスオブジェクトを使用してファイルへの書き込みや、区切りコンテンツのシリアル化を行えます。

ファイルライタ

ファイルライタ定義をイベントターゲットとして含む接続をパブリッシュすると、Process Designer もサービスを作成します。

FileWriteTask プロセスオブジェクトには、ターゲットファイルの名前、相対ファイルパス（イベントターゲットのベースディレクトリに基づく）、およびターゲットに書き込む文字列コンテンツが含まれます。

```
<FileWriteTask>

    <!-- file path, if required, which must be relative for FTP/SFTP -->
    <filePath> documents </filePath>
    <!-- target file name -->
    <fileName> test.txt</fileName>

    <!-- content format, which determines the applicability of other fields -->
    <format>PlainText|Binary|Attachment|JSON|XML</format>

    <!-- applies only if format is PlainText or Binary - if Binary, content is Base64-encoded string -->
    <content>Test file content</content>

    <!-- applies if content is XML or JSON, in which case, use object or objects -->
    <object>process object</object>
    <objects>a list of process objects</objects>

    <!-- optional, peovide the ObjectName and listName -->
    <objectName>order</objectName>
    <listName>orders</listName>

</FileWriteTask>
```

注: パス内のファイルまたはフォルダが存在しない場合は、自動的に作成されます。

FileInformation プロセスオブジェクトの詳細については、「ファイル情報プロセスオブジェクト」 (ページ 25)を参照してください。

区切りコンテンツ

区切りコンテンツイベントターゲットを含む接続をパブリッシュすると、Process Designer は、区切りファイルへのプロセスオブジェクトのシリアル化に使用できるサービスを作成します。

シリアル化は、区切りコンテンツプロセスオブジェクト（区切りコンテンツファイルの一般モデルを表す）とカスタムプロセスオブジェクトのどちらについても行えます。

カスタムプロセスオブジェクトのシリアル化

イベントターゲットの区切りファイルを処理するプロセスオブジェクトは、次のように動作します。

1. SerializeToDelimitedContentTask を使用して要求オブジェクトにアクセスします。
2. SerializeToDelimitedContentResult を使用して、シリアル化の結果にアクセスします。

以下に例を示します。

```
<SerializeToDelimitedContentTask>
   <!-- for FTP/SFTP only -->
   <host>127.0.0.1</host>

   <fileName> users2.csv </fileName>
   <filePath> CustomModelProcess </filePath>
   <delimiter> ; </delimiter>
   <skipHeaders> true </skipHeaders>
   <customObjects>
       <Email> bob@test.com </Email>
       <Password> 222222 </Password>
       <Phone_number> 333-3333-5554 </Phone_number>
```

```
        <User_name> Bob </User_name>
    </customObjects>
    <customObjects>
        <Email> bill@test.com </Email>
        <Password> 3333 </Password>
        <Phone_number> 444-222-111 </Phone_number>
        <User_name> Bill </User_name> </customObjects>
    <header>
        <name> User_name </name>
        <fieldIndex> 1 </fieldIndex>
    </header>
    <header>
        <name> Phone_number </name>
        <fieldIndex> 2 </fieldIndex>
    </header>
    <header>
        <name> Password </name>
        <fieldIndex> 3 </fieldIndex>
    </header>
</SerializeToDelimitedContentTask>
```

要求には、ターゲットファイル名、相対ファイルパス、カスタムオブジェクトの一覧が含まれている必要があります。

注: イベントターゲットのプロパティで設定されているデフォルト値を上書きする必要がある場合は、区切り文字とテキスト修飾子文字を使用できます。例えば、プロセスオブジェクトを 1 つずつ生成する場合は、skipHeaders 属性を上書きできます。その場合、最初のレコードにヘッダーを書き込んだ後、ファイル内の他のすべてのレコードを追加するときにはヘッダーをスキップできます。

ヘッダーの処理

Process Designer は、カスタムプロセスオブジェクトを使用する際、単純なオブジェクトは自動的にシリアル化しますが、複雑なフィールド（参照やオブジェクトリスト）はスキップします。生成されたファイルには、最初のプロセスオブジェクトの単純なフィールド名を使用したヘッダーの一覧が含まれています。この方法は、便利なときもありますが、デメリットもいくつかあります。

- プロセスオブジェクトのフィールドが特定の順序に依存していないため、結果ファイルでフィールド順序が定義されません。
- 最初のオブジェクトにオプションのフィールドがない場合、これらのフィールドは、他のすべてのオブジェクトにあっても無視されます。
- 不要なフィールドをスキップすることはできません。

これらのデメリットは、要求オブジェクトに一連のカスタムヘッダーを含めることで解消できます。Process Designer はこれらのヘッダーを使用して、指定したフィールドのみを含む区切りレコードを指定した順序で生成します。

注: カスタムヘッダーは、カスタムプロセスオブジェクトをシリアル化する場合のみ使用します。組み込み区切りコンテンツプロセスオブジェクトをシリアル化する場合、ヘッダーに関する情報はすでに含まれているため、カスタムヘッダーを使用する必要はありません。

シリアル化の結果

どの手法を使用する場合でも、各ファイルの処理後には次の結果が表示されます。

- 処理済みレコード数とターゲットファイルに正常に書き込まれたレコード数を示す 2 つのカウンタ。
- 変更日、パス、ファイル名、拡張子、生成されたファイルのサイズ。
- 操作のステータス。
- メッセージ文字列（オプション）。

以下に例を示します。

```
<SerializeToDelimitedContentResult>
   <message/>
   <processedRecordsCount> 10 </processedRecordsCount>
   <writtenRecordsCount> 8 </writtenRecordsCount>
   <success> true </success>
   <fileInfo> <lastModified> 2015-05-15T14:07:11.475Z </lastModified>
       <dir> D:/MyDirectory/DelimitedFiles/CustomModelProcess </dir>
       <name> users2 </ns7:name>
       <path> D:/MyDirectory/DelimitedFiles/CustomModelProcess/users2.csv </path>
       <fullName> users2.csv </fullName>
       <ext> csv </ext>
       <size> 116 </size>
   </fileInfo>
</SerializeToDelimitedContentResult>
```

第 6 章

ファイルコネクタを使用したプロセス定義

この章では、以下の項目について説明します。

Process Design の考慮事項

このコネクタを使用してプロセスを設計する際は、次の点に注意します。

- ファイルコンテンツを保持する入力フィールドは、接続プロパティで定義されます。
- プロセスは Secure Agent で実行する必要があります。
- プロセスを保存してパブリッシュした後は、接続プロパティで使用する各イベントソースまたはイベントターゲットを有効にし、プロセスで使用できるようにする必要もあります。

他の考慮事項は、扱うアプリケーション統合に応じて変わります。詳細は、次に説明するプロセスの例を参照してください。

ファイルコネクタを使用した処理の例

次の例は、電話番号の一覧を作成してレコード数をクライアントにパブリッシュするため、ファイルコネクタを使用して区切りコンテンツを読み取り、プロセスオブジェクトの一覧に変換し、サービスを呼び出す方法を示しています。

最初に接続を設定します。

1. 基本接続プロパティを定義し、接続タイプとして［ファイル］を選択します。このとき、接続を実行するエージェントを選択します。
2. ［イベントソース］タブで、［区切りコンテンツライタ］イベントを追加します。
3. 名前を入力します。パブリッシュ直後に接続を使用できるようにするには、イベントソースを有効にします。接続をすぐに使用できるようにしなくてもよい場合は、無効にできます。
4. 監視対象のディレクトリを指定します。このフィールドは必須です。他のオプションのプロパティを指定して、特定のファイルやサブディレクトリを除外したり含めたりすることもできます。

5. 必要に応じて他の解析オプションを入力します。
6. ［組み込みプロセスオブジェクトの使用］に対しては［いいえ］を選択します。［カスタムオブジェクト］については、読み取り中の区切りファイルヘッダー内のヘッダー名と完全一致する、ヘッダー名のカンマ区切りリストを入力します。以下に例を示します。

   ```
   Name,Street,City,State,PostalCode,Country,Phone
   ```
7. オプションの［ファイル読み取りロックの設定］を必要に応じて選択します。［その他の属性］フィールドでは、ファイルコネクタでは公開されていないが、Apache Camel ファイルリスナでサポートされる他のパラメータを指定できます。
8. プロセスを保存、テスト、およびパブリッシュします。

次にプロセスを作成します。

1. プロセスを作成します。接続で定義されたのと同じ Secure Agent 上で実行されるようにします。
2. ［開始］タブで DelimitedContent という接続を選択し、接続が監視対象ディレクトリで新しいファイルを検出したら、ファイルコンテンツを持つことになる入力フィールドを自動的に使用できるようにします。
3. プロセスのプロパティで 2 つの一時変数を作成し、ファイル内の区切りコンテンツに対して反復できるようにします。
   a. 1 つはレコードのフルセットを保持します。例えば、TempIterator です。
   b. もう 1 つはファイル内で反復する際の各オブジェクトを保持します。例えば、TempCurrent です。
4. 割り当てステップを作成し、ファイルレコード（プロセスオブジェクトリスト）を TempIterator フィールドに割り当てます。
5. 割り当てステップを作成し、リストの現在のレコードを一時変数 TempCurrent に取り込みます。TempCurrent のソースとして計算式を指定し、次を使用して最初のレコードを取得します。

   ```
   list:head($temp.TempIterator)
   ```
6. ディシジョンステップを追加して、レコードが設定されたことを検証します。
7. 設定済みブランチで、データの処理に必要な関数を実行します。この例は、RequestBin へのサービス呼び出しを作成し、ファイルから読み取った各レコードを渡し、電話番号の一覧を生成します。
8. 割り当てステップを追加して、プロセスオブジェクトの一覧から処理済みレコードを削除します。TempIterator のソースとして計算式を指定し、次を使用します。

   ```
   list:remove($temp.TempIterator,1)
   ```
9. TempCurrent を取り込む割り当てステップにループバックするジャンプステップを追加し、次のレコードをプロセスオブジェクトから取得します。
10. 設定解除ブランチでは、この例は、レコード数をクライアントにパブリッシュし、プロセスオブジェクトから FileInfo を読み取るサービスを呼び出します。
11. プロセスを終了します。

索引

ふ