pitney bowes

Big Data Quality SDK

バージョン 12.0

Big Data Quality SDK ガイド

目次

1 - はじめに

このセクションの構成

はじめに

Big Data Quality SDKでは、Hadoop プラットフォームでデータ品質の操作を行うために MapReduce ジョブ、 Spark ジョブおよび Hive ユーザ定義関数を作成、設定、および 実行できます。

この SDK を使用すると、Hadoop プラットフォームで直接ジョブを作成および実行できるため、ネットワーク遅延をなくし、クラスタで分散されたデータ品質プロセスを実行することにより、パフォーマンスを大幅に向上させることが可能になります。

Big Data Quality SDKでは、次のモジュールがサポートされています。

1. Advanced Matching モジュール
2. Data Normalization モジュール
3. Universal Name モジュール
4. Universal Addressing モジュール

SDK の使用
この SDK は現在、次を介して使用できます。

1. Java API: MapReduce と Spark をサポート
2. Hive ユーザ定義関数

レポート

Big Data Quality SDKでは、特定のジョブについて*レポート*機能を使用できます 。この機能は、サポートされる各機能に対する特定のカウンタを使用して、 対応するジョブにおいて達成されたマッチの成功状況を監視できるようにします。さまざまなカウンタにより、 実行されたジョブの重複レコードの数やユニーク レコードの数、その他の役に立つパラメータが追跡されます 。

*レポート*機能は現在、次のジョブでサポートされています。

- Interflow Match
- Intraflow Match
- Transactional Match
- Open Name Parser
- Validate Address
- Validate Address Global
- Validate Address Loqate

ワークフロー

SDK を使用するには、次のコンポーネントが必要です。

Big Data Quality SDKのインストール
Big Data Quality SDKの JAR ファイルをシステムにインストールし、アプリケーションで使用できるようにする必要があります。

クライアント アプリケーション
SDK を使用して必要なデータ品質操作を呼び出して実行するために作成する必要がある Java アプリケーション。Big Data Quality SDKの JAR ファイルを Java アプリケーションにインポートする必要があります。

Hadoop プラットフォーム
Big Data Quality SDKを使用してジョブを実行する際、まず、設定済みの Hadoop プラットフォームからデータが読み込まれ、関連する処理が実行された後、出力データが Hadoop プラットフォームに書き出されます。

このため、使用するマシンで Hadoop のアクセスの詳細情報を正しく設定しておく必要があります。詳細については、**概要**（8ページ）を参照してください。

リファレンス データ
Big Data Quality SDKで必要なリファレンス データは、Hadoop クラスタに配置されます。

Java API
Java API を使用する場合は、次のいずれかの場所にリファレンス データを配置できます。

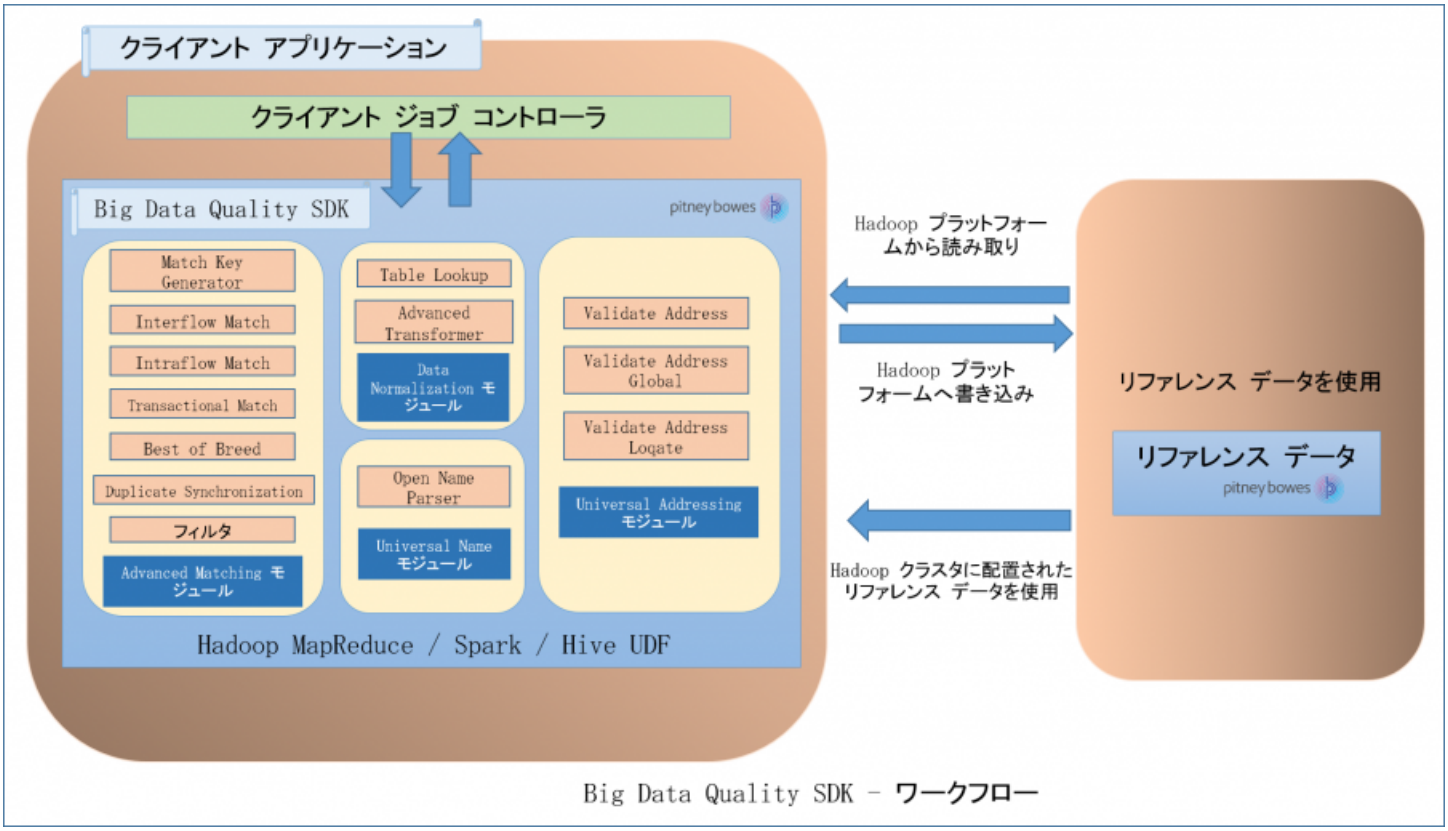
- **ローカル データ ノード**: リファレンス データはクラスタ内の使用可能な全てのノードに配置されます。

  注 : これはフェイルセーフな方法とはいえません。

- **Hadoop Distributed File System (HDFS)**: リファレンス データは HDFS ディレクトリに配置されます。この方法では、データがフェイルセーフに保管されます。

Hive UDF
Hive UDF を使用する場合は、リファレンス データをクラスタの各ローカル データ ノードに配置する必要があります。

注 : また、この SDK では、パフォーマンス向上のための分散キャッシュを実行できます。



Big Data Quality SDK – ワークフロー

この SDK の対象ユーザ

Big Data Quality SDKは、次のようなユーザを対象としています。

1. Hadoop 上のデータに対してデータ品質処理を実行したいと考えている顧客。
2. MapReduce または Spark のプログラミングに精通しており、特定の使用事例に関するソリューションを作成したいと考えている Hadoop 開発者。
3. 既存のデータに対してデータ クレンジング、*データ拡張*、*データの重複排除*、*データ統合*の操作を実行したいと考えている Hadoop 開発者。
4. MapReduce または Spark の複雑な部分には精通していないが、SQL と構文的に似ている Hive Query Language (HQL) は抵抗なく使用できる Hive ユーザ。

2 - インストール

このセクションの構成

システム要件

Hadoop Distributed File System (HDFS) を使用:

1. Java JDK 1.7 以降
2. Hadoop 2.6 以降
3. Spark 2.0.1 以降

Hive を使用:

1. Hive 1.2
2. 任意の Hive クライアント(例えば、Beeline など)

注： Spectrum™ Technology Platformは、Hadoop クラスタでのみ実行できます。

必要なオペレーティング システムの更新

Big Data Quality SDKをインストールする前に、お使いのオペレーティング システムに対して提供されているすべての最新製品アップデートを必ず適用してください。特に Java に関連する問題の修正は必須です。

SDK のインストール

概要

ウェルカム メールに記載されているリンクを使用して、ZIP ファイルをダウンロードします。`BigDataSDK120F0101.zip`といった名前の、一般的なインストーラ ZIP ファイルがダウンロードされます。

ダウンロードした ZIP ファイルのコンテンツをコンピュータ上に解凍し、インストーラを実行して、その指示に従ってインストール プロセスを進めます。インストールが完了すると、SDK ツールがシステム上に追加され、定義された場所に配置されます。

これで、Big Data Quality SDK JAR ファイルをプロジェクトにインポートし、お使いのコンピュータから API にアクセスことが可能です。

サポートされるモジュール

Big Data Quality SDKは、以下のモジュールをサポートしています。

1. Advanced Matching モジュール
2. Data Normalization モジュール
3. Universal Name モジュール
4. Universal Addressing モジュール

> **注 :** Universal Addressing モジュールの最初の *Validate Address* ジョブを作成する前に、Acushare サービスを起動しておく必要があります。詳細については、**Acushare サービスの実行**（12ページ）を参照してください。

SDK の使用

SDK は現在、次を介して使用できます。

1. Java API
   - MapReduce API
   - Spark API
2. Hive ユーザ定義関数

インストーラに含まれるもの

SDK インストール ZIP ファイルには、次のコンポーネントが含まれています。

1. `Readme.txt`
2. `sdkinst.bin`: LINUX コンピュータ用のインストーラ。
3. `sdkinst.exe`: WINDOWS コンピュータ用のインストーラ。

Windows での SDK のインストール

Windows コンピュータ上に Big Data Quality SDKをインストールするには、以下の手順を実行します。

1. ウェルカム メールまたはリリース通知メールに記載されているダウンロード手順に従って、Big Data Quality SDK ZIP インストーラ ファイルをダウンロードします。
2. アーカイブからのすべてのファイルを、Big Data Quality SDKをインストールする場所に展開します。
3. インストール ディレクトリに移動して、*sdkinst.exe* という名前のインストーラを探します。
4. `sdkinst.exe`ファイルをダブルクリックします。インストール ウィザードが表示されます。
5. **[次へ]** をクリックします。**[インストール フォルダを選択]** ウィンドウが表示されます。

   ここで、Big Data Quality SDKをインストールするフォルダを指定できます。例えば、`C:\Program Files\Pitney Bowes\Spectrum BigDataSDK\SDK`です。

   a) **[選択]** ボタンをクリックして、必要なフォルダを選択します。
   b) デフォルト フォルダを選択する場合は、**[Restore Default Folder(デフォルト フォルダに戻す)]** ボタンをクリックします。

   重要： デフォルト以外のフォルダをインストール ディレクトリとして選択する場合は、絶対インストール パスの長さが 34 文字を超えていないことを確認してください。

   デフォルトのインストール パスは 27 文字で許容範囲内に収まっています。

```
/root/PBSpectrum_BigDataSDK
```

6. **[次へ]** をクリックします。

   [インストール前の注意事項] 画面で、インストール情報を確認します。
7. **[インストール]** をクリックします。Big Data Quality SDKがコンピュータ上にインストールされます。
8. **[完了]** をクリックしてインストール プロセスを終了します。
9. SDK が正しくセットアップされていることを確認します。SDK をインストールした場所に移動します。例えば、`C:\Program Files\Pitney Bowes\Spectrum BigDataSDK\SDK`です。

コンピュータ上に SDK が正しくインストールされると、次のフォルダがインストール ディレクトリに追加されます。

- `API`
- `Documentation`
- `modules`

- samples
- utilities

注： Data Normalization モジュール、Universal Name モジュールまたは Universal Addressing モジュール用の個別のリファレンス データをインストールする必要があります。

Linux での SDK のインストール

Linux コンピュータ上にコマンド ラインで Big Data Quality SDKをインストールするには、以下の手順を実行します。

1. ウェルカム メールまたはリリース通知メールに記載されているダウンロード手順に従って、Big Data Quality SDKをダウンロードします。
2. アーカイブからのすべてのファイルを、Big Data Quality SDKをインストールするサーバー上の場所に展開します。
3. この場所にディレクトリを変更します。
4. 以下のコマンドを入力して、ファイルに対するexecute権限を取得します。

   ```
   chmod a+x sdkinst.bin
   ```

5. 次のコマンドを実行します。

   ```
   ./sdkinst.bin
   ```

   コマンド プロンプトの指示に従います。

6. 入力を求められたら、SDK をインストールするディレクトリを指定します。

   例えば、 /home/hadoop/BDQ_InstallPathです。

   重要： デフォルト以外のフォルダをインストール ディレクトリとして選択する場合は、絶対インストール パスの長さが 34 文字を超えていないことを確認してください。

   デフォルトのインストール パスは 27 文字で許容範囲内に収まっています。

   ```
   /root/PBSpectrum_BigDataSDK
   ```

   インストール前の注意事項が表示されます。

7. 注意事項を確認し、ENTERキーを押してインストールを続行します。
8. インストール ログ ファイルを参照して、Big Data Quality SDKが正しくインストールされたことを確認します。
9. 確認を終えたら、ENTERキーを押してインストーラを終了します。

コンピュータ上に SDK が正しくインストールされると、次のフォルダがインストール ディレクトリに追加されます。

- `API`
- `Documentation`
- `modules`
- `samples`
- `utilities`

> **注：** Data Normalization モジュール、Universal Name モジュールまたは Universal Addressing モジュール用の個別のリファレンス データをインストールする必要があります。

Acushare サービスの実行

最初の *Validate Address* ジョブを作成および実行する前に、Hadoop/Spark クラスタの各ノードで Acushare サービスを実行する必要があります。

> **注：** これは、*Validate Address* ジョブの実行前に一度だけ実行する必要がある操作です。

クラスタの各ノードで、以下の操作を行います。

1. Acushare セットアップ スクリプト `sdkrts.bin`を Big Data Quality SDK インストール パスからノード上の任意の場所にコピーします。

   重要： SDK サーバーでは、Acushare セットアップ スクリプト `sdkrts.bin`は、`<BDQ SDK_InstallPath>/SDK/utilities/dbloader/aq/runtime/bin` にあります。

2. 管理者権限で、または root ユーザとして、ノードにログインします。
3. Acushare インストーラ スクリプト `sdkrts.bin`をコピーした場所に移動します。
4. 以下のコマンドを入力して、ファイルに対する`execute`権限を取得します。

   ```
   chmod a+x sdkrts.bin
   ```

5. インストーラ ファイルを実行し、入力の要求に従って操作します。

   ```
   ./sdkrts.bin
   ```

6. 入力が求められたら、ENTER キーを押してデフォルトの実行時パス `/root/slave_node`を選択するか、任意の絶対パスを入力します。

   重要： *Validate Address* ジョブを実行するには、Acushare の実行時パスがクラスタのすべてのノードで同一である必要があります。

   > **注：** 選択するパスは、指定した時点でノード上のその場所に存在する必要があります。

   Acushare サービスは、インストールが正常に完了すると自動的に起動します。

7. Acushare サービスは、ノード上で手動で起動することもできます。`<Acushare runtime path>/runtime`に移動し、スクリプト ファイル `startrts.sh` を引数 `<Acushare runtime path>/runtime` 付きで実行します。

Acushare サービスの停止 ノード上の Acushare サービスを停止するには、`<Acushare runtime path>/runtime`に移動し、スクリプト ファイル `stoprts.sh` を引数 `<Acushare runtime path>/runtime` 付きで実行します。

Acushare サービスのアンインストール Acushare サービスをノードからアンインストールするには、スクリプト ファイル `Uninstall_SDKRTS.sh`を`<Acushare runtime path>/Uninstall` から実行します。

リファレンス データ

リファレンス データの概要

Pitney Bowes リファレンス データは、データ品質を保証するためにシステム内の他のデータ フィールドによって使用される許容値の集合を定義します。これによって、データの妥当性、正確性、一貫性が高まります。データからさらなる価値を引き出し、ビッグ データ システムから信頼できるデータを取得することができるようになります。

例えば、Data Normalization モジュールでリファレンス データを使用すると、企業全体にわたって単一の顧客の同一性を確立することができます。顧客情報の適切な定義は、運用効率を改善するための第一歩です。

重要： *Validate Address* および *Validate Address Global* ジョブでは、Hadoop クラスタのすべてのデータ ノードにリファレンス データが配置されている必要があります。*Validate Address Loqate* ジョブでは、リファレンス データが 1 つのノードに配置され、さらにそのデータがその他すべてのデータノードにマウントされている必要があります。

インストール ディレクトリの構造

SDK インストール ディレクトリ下の `Utilities/dbloader`ディレクトリには、次の子フォルダがあります。

dataquality このフォルダには、以下のモジュールの参照データをインストールするための JAR およびスクリプトが含まれます。

- Data Normalization モジュール

- Universal Name モジュール

  注： 詳細については、リファレンス データの使用: **Data Normalization** モジュールおよび **Universal Name** モジュール（14ページ）を参照してください。

aq

次のファイルが含まれます。

- リファレンス データをインストールするための `scripts/server/installdb_unc.sh`スクリプト。このデータをインストールまたは展開するには、このスクリプトを実行する必要があります。
- Universal Addressing モジュールの *Validate Address* ジョブのための Acushare サービス セットアップ情報が含まれている `runtime` フォルダ。

  注： 詳細については、リファレンス データの使用: **Universal Addressing** モジュール（15ページ）を参照してください。

リファレンス データの使用: Data Normalization モジュールおよび Universal Name モジュール

Data Normalization モジュールおよび **Universal Name** モジュールでリファレンス データを使用するには、データ ローダー スクリプト ファイル (`installdb_dnm` など) を実行する必要があります。スクリプト ファイルを実行すると、リファレンス データがお使いのコンピュータ上に展開されます。

`installerdb_dnm`などのスクリプト ファイルと JAR ファイルが、同じフォルダに存在することを確認してください。

1. コンピュータにログインします。
2. SDK をインストールした場所にディレクトリを変更します。

   コンピュータ上に Big Data Quality SDKを正しくインストールすると、リファレンス データ ローダーがディレクトリ `BDQ_InstallPath/SDK/utilities/dbloader/unix/bin` に配置されるはずです。

3. リファレンス データ ローダー スクリプトを実行します(例: `installdb_dnm`)。
   ステージのリストが番号付きで表示され、ステージを選択するように求められます。
4. データをロードするステージに対応する番号を入力します。
5. ダウンロード後にリファレンス データ セットを展開して配置する場所のパスを指定します。

リファレンス データ入力は、Data Normalization モジュールと Universal Name モジュールのジョブの実行に必要な、Data Normalization モジュールの基本テーブルや Core Name データベースなどです。

6. 出力ディレクトリのパスを指定します。このパスに入力データが展開されます。
7. ログファイルを表示するかどうかを尋ねるプロンプトが表示されます。選択を適宜行います。
8. データのロードが開始します。データは、指定した出力ディレクトリに展開されます。

注： 各ステージに対して、この手順を繰り返します。

リファレンス データの使用: Universal Addressing モジュール

リファレンス データにアクセスして使用するには、最初にこのデータを eStore から ZIP 形式で入手する必要があります。

Validate Address Global と *Validate Address Loqate* の場合は、ZIP ファイルを解凍するだけでリファレンス データは使用可能になります。

Validate Address の場合は、上記の手順でリファレンス データをコンピュータに解凍します。

注： `execute`権限が`aq` フォルダに対して付与されていることを確認します。

1. 管理者権限で、または root ユーザとして、ログインします。
2. ディレクトリを `<BDQ_Installation>/SDK/utilities/dbloader/aq/scripts/server` に変更します。
3. 次のコマンドを使用して、スクリプト `installdb_unc`を実行します。

   ```
   sh installdb_unc.sh <BDQ_Installation/SDK> <Acushare runtime path>
   ```

   このコマンドでは、Acushare サービスが実行中かどうかも確認されます。実行中でなければ、このコマンドによってサービスが起動されます。

4. このコマンドを実行すると、以下のオプションが提示されます。
   - **US サブスクリプション**: 1 を押すと、次の手順に示すように、使用可能なデータ ロードのリストが表示されます。
   - **終了**: 99 を押すと終了します。
5. ロードするデータのタイプを表す特定の数値を入力します。

```
1. Subscription Database
2. Delivery Point Validation
3. Residential Delivery Indicator
4. Early Warning System
5. LACSLink Database
6. SuiteLink Database

99. Exit


Enter the number of the type of data you want to load
and then press enter:
```

6. 入手したデータ セットが置かれているパスを指定します。

   eStore から入手したデータは、Universal Addressing モジュールのジョブを実行するために必要なリファレンス データ入力として使用できます。出力ファイルの場所については、デフォルトの出力パスがシステムによって表示されます。

7. 入力ファイルと出力ファイルの場所が表示されます。

   そのまま続行する場合は c、デフォルト パスを変更する場合は m、終了する場合は q を入力します。

```
The Residential Delivery Indicator load environment is currently set to:

Residential Delivery Indicator input file location:
Residential Delivery Indicator output file location: /root/SDK/utilities/dbloader/addressquality/s

Enter c to (c)ontinue
   or m to (m)odify
   or q to (q)uit

   ===>
```

   入力データが、指定した出力ファイルの場所に解凍されます。

8. 新しい RDI ファイルの場所が正しいかどうかを確認するように求められます。y または n を入力します。

```
Please enter full path where you would like to install
the RDI file ==> /root/out

The new RDI file location will be:  /root/out
Is this correct?

Enter (y)es to continue.
      (n)o to try again.

   ===> y

The RDI file output location /root/out does not exist.
Do you want to create it now?

Enter (y)es to create the new RDI file area.
      (n)o to exit.

   ===>
```

データのロードが開始します。データは、指定した出力ディレクトリに展開されます。

注： ロードするデータのタイプごとにこの手順を繰り返します。

3 - モジュール

このセクションの構成

Advanced Matching モジュール

Advanced Matching モジュールは、複数の入力ファイル間や、複数の入力ファイル内でレコードを照合します。また、Advanced Matching モジュールでは、名前、住所、名前と住所、名前/住所以外のフィールド (社会保障番号、生年月日等) 等、さまざまなフィールドの照合も可能です。

このモジュールでは、適切な設定を使用している最適なレコードを選択する、特定のグループのすべてのレコードを同期する、レコード グループから特定のレコードをフィルタリングして除外する、といった方法でグループのレコードを統合する各種ジョブを使用できます。

サポートされるジョブ

Big Data Quality SDKの Advanced Matching モジュールでは、次のジョブがサポートされます。

1. Match Key Generator
2. Interflow Match
   - マッチ キーを生成する
   - グループ化オプションを通じて既存のマッチ キーを使用する
3. Intraflow Match
   - マッチ キーを生成する
   - グループ化オプションを通じて既存のマッチ キーを使用する
4. Transactional Match
   - マッチ キーを生成する
   - グループ化オプションを通じて既存のマッチ キーを使用する
5. Best of Breed
6. Duplicate Synchronization
7. フィルタ

   注： グループ化オプションを使用する際、入力ファイルにすでに存在しているマッチ キーを使用してグループ化操作が実行されます。

Match Key Generator

Match Key Generator は、レコードごとに非ユニーク キーを作成します。この非ユニーク キーは、潜在的な重複レコードのグループを特定するためにマッチング ステージで使用できます。マッチ キーを使用すると、レコードをマッチ キー別にグループ化し、各グループ内でのみレコードを比較できるので、マッチング プロセスが促進されます。

マッチ キーは、定義したルールを使用して作成され、入力フィールドから構成されます。指定する入力フィールドごとに、そのフィールドで実行されるアルゴリズムが選択されます。 その後、各アルゴリズムの結果を連結して、単一のマッチ キー フィールドが作成されます。

マッチ キーの作成に加え、後のデータフローの Intraflow Match ステージまたは Interflow Match ステージで使用する Express マッチ キーも作成できます。

複数のマッチ キーおよび Express マッチ キーを作成できます。

例えば、次のような入力レコードがあり、

名 - Fred
姓 - Mertz
郵便番号 - 21114-1687
性別コード - M

次のようなレコードのデータを組み合わせてマッチ キーを生成するマッチ キー ルールを定義したとします。

入力フィールド	開始位置	長さ
郵便番号	1	5
郵便番号	7	4
姓	1	5
名	1	5
性別コード	1	1

次のようなキーになります。

211141687MertzFredM

Interflow Match

Interflow Match は、2 つの入力レコード ストリーム内の類似するデータ レコード間でマッチを検出します。最初のレコード ストリームはサスペクト レコードのソースで、2 番目のストリームは候補レコードのソースです。

Interflow Match では、マッチ グループ条件 (マッチ キー等) を使用して、特定のサスペクト レコードと重複する可能性があるレコードのグループを識別します。

レポート

Interflow Match ジョブを使用して、ジョブの結果を監視することができます。使用可能なカウンタは次の通りです：

DUPLICATE_COLLECTIONS	コレクション番号によってグループ化されたサスペクト レコードとその重複レコードで構成される、重複コレクションの数。
EXPRESS_MATCHES	1 つのコレクションで作成された Express マッチの数。 Express マッチは、サスペクトと候補が指定されたフィールド内の内容に正確にマッチした場合に作成され、通常は ExpressMatchKey が Match Key Generator によって提供されます。Express マッチが作成された場合、サスペクトと候補の重複を判定するためのそれ以上の処理は行われません。
AVERAGE_SCORE	すべての重複の平均マッチ スコア。 有効な値は 0 ～ 100 です。0 は精度の低いマッチを意味し、100 は完全一致を意味します。
INPUT_SUSPECTS	マッチャーが他のレコードとの照合を試みた入力ストリーム内のレコードの数。
SUSPECTS_WITH_DUPLICATES	少なくとも 1 つの候補レコードと一致した入力サスペクトの数。
UNIQUE_SUSPECTS	どの候補レコードとも一致しなかった入力サスペクトの数。
SUSPECTS_WITH_CANDIDATES	マッチ グループ内に候補レコードが少なくとも 1 つある、つまり照合の試みが少なくとも 1 回は行われた入力サスペクトの数。
SUSPECTS_WITHOUT_CANDIDATES	マッチ グループ内に候補レコードがない、つまり照合の試みが行われなかった入力サスペクトの数。
TOTAL_DUPLICATE_CANDIDATES	検出された重複候補の総数。

TOTAL_DUPLICATE_SCORE	すべての重複の合計マッチ スコア。

Intraflow Match

Intraflow Match は、単一の入力ストリーム内の類似するデータ レコード間でマッチを検出します。データフローの他のステージで定義または作成されたフィールドに基づいて階層型のルールを作成できます。

レポート

Intraflow Match ジョブを使用して、ジョブの結果を監視することができます。使用可能なカウンタは次の通りです：

INPUT_RECORDS	マッチング ソート実行前のマッチング ステージにおけるレコードの数。
DUPLICATE_RECORDS	マッチ グループ内の重複レコード (サスペクト レコードまたは候補レコード) の数。
UNIQUE_RECORDS	各マッチ グループで他のレコードにマッチしないサスペクトまたは候補レコードの数。 マッチ グループ内に 1 つしか存在していないレコードであれば、サスペクトは自動的にユニーク レコードとなります。
MATCH_GROUPS	(グループ化) マッチ キーでグループ化されたレコード。
DUPLICATE_COLLECTIONS	コレクション番号によってグループ化されたサスペクトレコードとその重複レコードで構成される、重複コレクションの数。
EXPRESS_MATCHES	1 つのコレクションで作成された Express マッチの数。 Express マッチは、サスペクトと候補が指定されたフィールド内の内容に正確にマッチした場合に作成され、通常は ExpressMatchKey が Match Key Generator によって提供されます。Express マッチが作成された場合、サスペクトと候補の重複を判定するためのそれ以上の処理は行われません。
AVERAGE_SCORE	すべての重複の平均マッチ スコア。 有効な値は 0 ～ 100 です。0 は精度の低いマッチを意味し、100 は完全一致を意味します。
TOTAL_DUPLICATES	検出された重複の総数。
TOTAL_SCORE	すべての重複の合計マッチ スコア。

Transactional Match

Transactional Match は、重複を特定するため、サスペクト レコードと、あるグループのレコードを照合します。これらのレコードはまず、選択した列によりグループ化され、最初のレコードがサスペクト レコードとしてマークされます。グループの残りすべてのレコードは候補レコードと呼ばれ、サスペクト レコードと照合されます。

候補レコードが重複の場合は、コレクション番号が割り当てられ、そのマッチ レコード タイプに重複が設定され、その候補レコードが書き出されます。グループ内のマッチしない候補にはコレクション番号 0 が割り当てられ、そのラベルにユニークが設定され、その候補が書き出されます。

レポート

Transactional Match ジョブを使用して、ジョブの結果を監視することができます。使用可能なカウンタは次の通りです：

AVERAGE_SCORE	すべての重複の平均マッチ スコア。 有効な値は 0 ～ 100 です。0 は精度の低いマッチを意味し、100 は完全一致を意味します。
INPUT_SUSPECTS	マッチャーが他のレコードとの照合を試みた入力ストリーム内のレコードの数。
SUSPECTS_WITH_DUPLICATES	少なくとも 1 つの候補レコードと一致した入力サスペクトの数。
UNIQUE_SUSPECTS	どの候補レコードとも一致しなかった入力サスペクトの数。
SUSPECTS_WITH_CANDIDATES	マッチ グループ内に候補レコードが少なくとも 1 つある、つまり照合の試みが少なくとも 1 回は行われた入力サスペクトの数。
SUSPECTS_WITHOUT_CANDIDATES	マッチ グループ内に候補レコードがない、つまり照合の試みが行われなかった入力サスペクトの数。
TOTAL_DUPLICATES_SCORE	すべての重複の合計マッチ スコア。
TOTAL_DUPLICATES	検出された重複の総数。

Best of Breed

Best of Breed は、重複レコードのコレクションから選択する最良のデータを使用して新しい統合レコードを作成することで、重複レコードを統合します。この "スーパー" レコードは、最良の組

み合わせレコードと呼ばれます。処理対象レコードの選択で使用するルールを定義します。処理が完了すると、最良の組み合わせレコードがシステムに保持されます。

Duplicate Synchronization

Duplicate Synchronization は、レコードのコレクションから、そのコレクション内のすべてのレコードの対応するフィールドにコピーするフィールドを指定します。フィールド データをコレクション内の別のレコードにコピーするときに従う必要があるルールを指定できます。処理が完了すると、コレクション内のレコードがすべて保持されます。

Filter

Filter ステージでは、指定したルールに基づいて、レコードをレコードのグループに保持または削除します。

Data Normalization モジュール

Data Normalization モジュールでは、レコード内の語を調べ、その語が好ましい形式であるかどうかを確認します。

- **Table Lookup** — このステージは、語を評価し、その語とその語の妥当性確認済みの形式とを比較します。語が適切な形式でない場合は、その語を標準バージョンに置き換えます。Table Lookup には、単語のフルスペルを省略形に変更する、単語の省略形をフルスペルに変更する、ニックネームを正しい名前に変更する、ミススペルを訂正する機能等があります。
- **Advanced Transformer** — このステージは、一連のデータをスキャンして分割し、抽出データと非抽出データを既存のフィールドまたは新しいフィールドに配置します。

サポートされるジョブ

Big Data Quality SDKの Data Normalization モジュールでは、次のジョブがサポートされます。

1. Table Lookup
   - 正規化オプションを使用する Table Lookup
   - 特定オプションを使用する Table Lookup

- 分類オプションを使用する Table Lookup

2. Advanced Transformer
   - テーブルデータでの抽出オプションを使用する Advanced Transformer
   - 正規表現での抽出オプションを使用する Advanced Transformer

Table Lookup

Table Lookup ステージは、語とその語の妥当性確認済みの形式とを照合して正規化し、正規化したバージョンを適用します。この評価は、正規化する語をテーブルから検索して実行されます。

Advanced Transformer

Advanced Transformer ジョブは、テーブルまたは正規表現を使用して、一連のデータ列をスキャンして複数のフィールドに分割します。このコンポーネントは、特定の語、あるいは語の右側または左側から指定した数だけ単語を抽出します。抽出データと非抽出データを既存のフィールドまたは新しいフィールドに配置できます。

例えば、次の住所フィールドから一組の情報を抽出し、その情報を別のフィールドに配置するとします。

2300 BIRCH RD STE 100

これを行うには、語 STE と語 STE の右側にあるすべての単語を抽出する Advanced Transformer を作成することができます、分離されるフィールド：

2300 BIRCH RD

Universal Addressing モジュール

Universal Addressing モジュールは、住所品質モジュールで、住所の正規化とバリデーションを実行して、郵便物の配達品質を高めることができます。Universal Addressing モジュールを使用すると、住所データに対して郵便当局が定める品質規格への準拠を徹底できます。住所がこれらの規格に準拠していれば、郵便物を規定の配達日数でより確実に配達できます。また、差出人も、これらの規格に準拠すれば、郵便料金の大幅な割引を受けることができます。米国における郵便料金の割引については、**www.usps.com** にある USPS Domestic Mail Manual (DMM) を参照してください。

注： UAM ジョブの場合、リファレンス データは、クラスタ内のローカル データ ノードのみに配置する必要があります。

サポートされるジョブ

Big Data Quality SDKの Universal Addressing モジュールでは、次のジョブがサポートされます。

1. Validate Address

   注： 現在、このジョブでサポートされているのは、米国の住所検証のみです。

2. Validate Address Global
3. Validate Address Loqate

Validate Address

Validate Address は、郵便当局の住所データを使用して、住所を正規化し、妥当性を確認します。Validate Address は、情報を修正し、管轄の郵便当局が推奨する書式で住所の書式を整えることができます。また、郵便番号、都市名、州または省名など、欠落している郵便情報を追加します。

Validate Address は、Validate Address が住所の妥当性を確認したかどうか、返した住所の確信レベル、住所の妥当性が確認できなかった場合はその理由など、バリデーション処理に関する結果インジケータも返します。

Validate Address は、住所のマッチングと正規化において、住所行をコンポーネントに分割し、それらを Universal Addressing モジュールの各種データベースの内容と比較します。マッチを検出した場合、入力住所をデータベース情報に合わせて*正規化*します。データベースにマッチしなかった場合、Validate Address は、オプションで入力住所の*書式を整えます*。書式設定プロセスでは、該当する郵便当局の規則に従って住所行の構成を試みます。

注： 現在、Validate Address では米国住所のみがサポートされています。

CASS レポート

Big Data Quality SDK を使用して、CASS Certified™ モードで Validate Address ジョブを作成して実行できます。

また、次のタイプの CASS レポートを生成することもできます。

1. CASS レポート 3553
2. CASS 詳細レポート

3. Validate Address サマリ レポート

CASS 認定処理

また、CASS 認定™処理では USPS CASS 詳細レポートも生成されます。このレポートに含まれる情報は 3553 レポートと同じものですが、DPV、LACS、および SuiteLink に関する大幅に詳しい統計情報が含まれます。USPS CASS 詳細レポートは、郵便料金の値引きを受けるために必ずしも必要ではなく、郵便物と一緒に提出する必要はありません。

CASS 詳細レポートは次の 3 つのパートで構成されます。

1. *CASS 詳細*
2. *CASS 詳細 2*
3. *CASS 詳細 3*

SDK を使用しているときの CASS 設定の詳細については、**Validate Address MapReduce ジョブの使用**（117ページ）と**Validate Address Spark ジョブの使用**（119ページ）を参照してください。レポートの使用方法については、『*Dataflow Designer ガイド*』を参照してください。

CASS 3553 レポート

USPS から値引きを受けるには、USPS CASS 3553 レポートを郵便物と一緒に渡す必要があります。レポートには CASS 処理に使用したソフトウェアに関する情報、名前と住所のリストに関する情報、出力ファイルに関する情報、差出人に関する情報、およびその他の郵便物に関する統計が含まれます。USPS Form 3553 の詳細については、**www.usps.com** を参照してください。

レポートの使用方法については、『*Dataflow Designer ガイド*』を参照してください。

CASS 詳細レポート

USPS CASS 詳細レポートは、何かの値引きを受けるための条件として必ずしも USPS に提出する必要はありません。このレポートに含まれる情報の一部は 3553 レポートと同じものですが、DPV、LACS、および SuiteLink に関する大幅に詳しい統計情報が含まれます。

レポートの使用方法については、『*Dataflow Designer ガイド*』を参照してください。

Validate Address サマリ レポート

Validate Address サマリ レポートには、処理されたレコードの合計数や、検証された住所の数など、ジョブに関する統計が一覧表示されます。

レポートの使用方法については、『*Dataflow Designer ガイド*』を参照してください。

Validate Address Global

Validate Address Global は、米国およびカナダ以外のアドレスのアドレス標準化と検証機能が強化されています。Validate Address Global は、米国およびカナダの住所の妥当性も確認できますが、その他の国の住所の妥当性を確認する能力に優れています。米国およびカナダ以外の住所を大量に処理する場合は、Validate Address Global の使用を検討してください。

Validate Address Global は Universal Addressing モジュールの一部です。

Validate Address Global は、パーシング、検証、書式設定など、いくつもの手順を実行して、住所の品質を高めています。

住所のパーシング、書式設定、および正規化

住所データのフィールド入力の誤りを再構成することは、特に他国の住所で行う場合、複雑で難しい作業です。住所データをコンピュータのシステムに入力する際、曖昧になってしまう部分が多いからです。特に問題なのが、(企業や個人名をストリートの住所フィールドに入力するなど) 要素を誤ったフィールドに入力したり、省略形を使用する場合に、言語固有だけでなく、国固有の省略形に変えてしまうケースです。Validate Address Global は住所行の住所要素を識別し、正しいフィールドに割り当てます。これは実際の検証前に行う重要な作業です。再構成を行わなければ、"一致が見つからない" という結果になる可能性があります。

住所要素の正しい識別は、特定のフィールド長要件に合わせて住所を切り捨てたり、短縮しなければならない場合にも重要です。正しい情報が正しいフィールドに割り当てられていれば、特定の切り捨てルールを適用することができます。

- 住所行をパースおよび解析し、個々の住所要素を識別
- 30 を越える文字セットを処理
- 宛先国の郵便ルールに従って住所の書式を整える
- 住所要素を正規化 (AVENUE を AVE に変更するなど)

Global Address 検証

住所の検証は、正しくパースされた住所データを郵便組織または他のデータ プロバイダが提供する参照データベースと比較する訂正処理です。Validate Address Global は、洗練されたファジー マッチング テクノロジーを使用して個々の住所要素を検証し、正しいことを確認するとともに、郵便規格とユーザの優先設定に基づいて出力を正規化および書式設定します。FastCompletion 検証タイプは、簡易住所入力アプリケーションに使用できます。いくつかの住所フィールドには切り捨てられたデータを入力することができ、この入力に基づいて提案を生成します。

住所を完全に検証できない場合もあります。Validate Address Global には、配達可能性によって住所を分類する、ユニークな配達可能性評価機能があります。

レポート カウンタ

Validate Address Global ジョブでは、実行が完了した後のジョブの統計情報を監視できます。各カウンタは、特定の Validate Address Global ジョブが実行されているすべてのサポート対象国にわたる統計情報を提供します。

サポートされている国のリストについては、**#unique_39**を参照してください。

国ベースのカウンタ

これらのカウンタは、さまざまなサポート対象国に対するレポート統計を提供します。それぞれの国ラベルは、カウンタ値に対応する国コードで始まります。

例えば、以下のカウンタは、米国のレポート統計を示しています。

1. UNITEDSTATES_STATUS_I4_COUNT
2. UNITEDSTATES_STATUS_S_COUNT
3. UNITEDSTATES_STATUS_I3_COUNT
4. UNITEDSTATES_FAILED_COUNT
5. UNITEDSTATES_STATUS_I2_COUNT
6. UNITEDSTATES_STATUS_C_COUNT
7. UNITEDSTATES_STATUS_V_COUNT

同様に、この Validate Address Global ジョブが実行されているすべてのサポート対象国についても同じカウンタのリストが存在します。

サマリ カウンタ

サマリ カウンタは、国々における特定の国タイプの値の和を示します。

例えば、`SUMMARY_FAILED_COUNT`は、特定の Validate Address Global ジョブが実行されているすべてのサポート対象国についての `FAILED_COUNT` カウンタの値の和です。

1. SUMMARY_STATUS_I4_COUNT
2. SUMMARY_STATUS_I2_COUNT
3. SUMMARY_END_TIME
4. SUMMARY_START_TIME
5. SUMMARY_STATUS_V_COUNT
6. SUMMARY_STATUS_C_COUNT
7. SUMMARY_CHARSET
8. SUMMARY_DEFAULT_COUNTRY
9. SUMMARY_STATUS_I3_COUNT
10. SUMMARY_STATUS_S_COUNT
11. SUMMARY_FAILED_COUNT
12. COUNTRY: 住所検証が実行されている国コードのカンマ区切りリスト。

13. SUMMARY_CASING: 出力の大文字/小文字を区別する方法。詳細については、『*Addressing ガイド*』の「*Validate Address Global*」の「*オプション*」セクションを参照してください。

Validate Address Loqate

Validate Address Loqate は、郵便当局の住所データを使用して、住所を正規化し、妥当性を確認します。Validate Address Loqate は、情報を修正し、管轄の郵便当局が推奨する書式で住所の書式を整えることができます。また、郵便番号、都市名、州/省名など、欠落している郵便情報を追加します。

Validate Address Loqate は、Validate Address Loqate が住所の妥当性を確認したかどうか、返された住所の確信レベル、住所の妥当性が確認できなかった場合はその理由など、検証処理に関する結果インジケータも返します。

Validate Address Loqate は、住所のマッチングと正規化において、住所行をコンポーネントに分割し、それらを Universal Addressing モジュールの各種データベースの内容と比較します。マッチを検出した場合、入力住所をデータベース情報に合わせて正規化します。データベースにマッチしなかった場合、Validate Address Loqate は、オプションで入力住所の書式を整えます。書式設定プロセスでは、該当する郵便当局の規則に従って住所行の構成を試みます。
ValidateAddressLoqate は、Universal Addressing モジュールに含まれています。

レポート カウンタ

Validate Address Loqate ジョブを使用して、ジョブの結果を監視することができます。使用可能なカウンタは次の通りです：

1. 住所の一致によって確認された元の郵便番号
2. マッチングに成功したレコードの総数
3. 家の不一致
4. 住所検証が試みられたレコードの総数
5. 入力レコード数
6. 数値範囲の不一致
7. 入力で有効なレコードの総数
8. 使用可能な郵便番号なし
9. 記録された非マッチ合計数
10. 訂正された総数
11. マッチしないレコードの総数
12. 住所一致によって訂正された郵便番号
13. 正常に返された標準住所
14. 処理された住所レコード

15. ストリートの不一致
16. 保持された元の郵便番号
17. Loqate によって処理されたレコード

Universal Name モジュール

正規化をきわめて正確に実行するには、一連のデータ列を複数のフィールドに分割する必要がある場合があります。Big Data Quality SDKは、個人名、企業名、およびその他多くの語や略語をパースする高度なパーシング機能を備えています。

サポートされるジョブ

Big Data Quality SDKの Universal Name モジュールでは、次のジョブがサポートされます。

1. Open Name Parser

Open Name Parser

OpenNameParser は、名前データ フィールドにある個人名、企業名、またはその他の名称を構成要素に分解します。パースされたこれらの名前要素は、名前のマッチング、名前の正規化、複数レコード名の統合など、他の自動化処理に使用できます。

レポート

Open Name Parser では、入力レコードの合計数や、名前データが含まれないレコードの合計数など、ジョブに関するサマリ統計を提供します。

一般的な結果

INPUT_RECORDS	入力内のレコード数。
NO_NAME_DATA_RECORDS	入力内の、パースする名前データを含まないレコードの数。
NAMES_PARSED_OUT	入力内の、パースされた名前の数。
LOWEST_NAME_PARSING_SCORE	入力内の名前に与えられたパーシング スコアの最小値。
HIGHEST_NAME_PARSING_SCORE	入力内の名前に与えられたパーシング スコアの最大値。

AVERAGE_NAME_PARSING_SCORE	入力内のパースされたすべての名前に与えられたパーシング スコアの平均値。

個人名のパーシング結果

PERSONAL_NAME_RECORDS	入力内の個人名の数。
CONJOINED_NAMES_PARSED	結合名を含むレコードからパースされた名前の数。 例えば、入力に、2 つの結合名を持つレコードが 5 つ、3 つの結合名を持つレコードが 7 つある場合、このフィールドのカウンタ値は、(5 x 2) + (7 x 3) = 31 になります。
TWO_CONJOINED_NAMES_RECORDS	結合名を 2 つ含む入力レコードの数。
THREE_CONJOINED_NAMES_RECORDS	結合名を 3 つ含む入力レコードの数。
TITLE_OF_RESPECT_NAMES	パースされた名前のうち、敬称を含む名前の数。
MATURITY_SUFFIX_NAMES	パースされた名前のうち、世代接尾語を含む名前の数。
GENERAL_SUFFIX_NAMES	パースされた名前のうち、一般接尾語を含む名前の数。
ACCOUNT_DESCRIPTION_PERSONAL_NAMES	パースされた名前のうち、アカウント説明を含む名前の数。
TOTAL_REVERSE_ORDER_NAMES	パースされた名前のうち、逆順序の名前の数 (出力フィールド `IsReverseOrder` が "True")。

企業名のパーシング結果

BUSINESS_NAME_RECORDS	企業名を含む入力レコードの数。
FIRM_SUFFIX_NAMES	パースされた名前のうち、企業接尾語を含む名前の数。
ACCOUNT_DESCRIPTION_BUSINESS_NAMES	入力レコードのうち、アカウント説明を含む名前の数。
TOTAL_DBA_RECORDS	略語 Doing Business As (DBA) を含み、出力フィールド `isPersonal` と `isFirm` の両方が "True" となる入力レコードの数。
TOTAL_PARSED	パースされた名前の総数。
TOTAL_NAME_PARSING_SCORE	すべての名前の合計パーシング スコア。

4 - Java API

このセクションの構成

はじめに

Java クラスは、あるタイプのすべてのインスタンスに共通する変数およびメソッドを定義する設計図またはプロトタイプです。Java クラスは、特定の種類のインスタンスの実装も定義します。

Java オブジェクトは、Java クラスのインスタンスです。これは Java クラスのリアルタイムのインスタンスであり、Java 仮想マシン (JVM) を用いて作成されます。クラスのインスタンスは、変数を用いて処理され、そのクラスのリアルタイム情報をカプセル化します。

クラスのメソッドは、あるクラスやそのオブジェクトが実行するさまざまな機能を定義します。メソッドは、C などの手続き言語における関数またはプロシージャに似ています。

パラメータは、オブジェクトがあるタスクを実行するのに必要とする情報を渡すために使用されます。

Java ソフトウェア オブジェクトは、メッセージを使用して互いにやりとりしたり、通信を行います。

Java テクノロジーの詳細については、**www.oracle.com/java** を参照してください。

SDK Java API のコンポーネント

Java API を使って Big Data Quality SDKジョブを実行するための主要コンポーネントは、以下のとおりです。

JAR ファイル

1. Hadoop JAR ファイル。
2. 以下の表に示された、必要な Big Data Quality SDKジョブが属するモジュールの JAR ファイル。

モジュール	ジョブ	JAR ファイル
Advanced Matching モジュール	すべての AMM ジョブ	*amm.core-12.0.jar*
Data Normalization モジュール	すべての DNM ジョブ	*dnm.core-12.0.jar*
Universal Addressing モジュール	Validate Address	*uam-universaladdress.core-12.0.jar*

モジュール	ジョブ	JAR ファイル
Universal Addressing モジュール	Validate Address Global	*uam-global.core-12.0.jar*
Universal Addressing モジュール	Validate Address Loqate	*uam-loqate.core-12.0.jar*
Universal Name モジュール	すべての UNM ジョブ	*unm.core-12.0.jar*

設定ファイル
マッチ ルール、入力ファイル詳細情報、出力ファイル詳細情報、MapReduce または Spark 詳細設定など、ジョブの実行に必要なすべてのパラメータと値を含む、XML 形式のファイル。

XML 設定ファイルのサンプルは、`<Big Data Quality bundle>\samples\configuration`にあります。

クライアント Java アプリケーション
API を使用して、Java API で提供される必要な Big Data Quality SDKジョブを作成および実行する Java アプリケーション。

Hadoop プラットフォーム
作成されたジョブは、入力データにアクセスし、出力データをファイルに書き出すために、設定済みの Hadoop プラットフォームにアクセスします。

SDK の使用

SDK は、次の 2 つのいずれかの方法で Big Data Quality SDKジョブの実行に使用できます。

1. コンソール上で、モジュール固有の JAR ファイルを直接実行し、XML 形式の各種設定プロパティ ファイルをコマンドの引数として渡します。

   MapReduce ジョブの場合は `hadoop`コマンド、Spark ジョブの場合は `submit-spark` コマンドを実行します。

   手順については、**設定プロパティ ファイルの使用**（36ページ）を参照してください。

2. 関連する Big Data Quality SDKモジュールの JAR ファイルをインポートすることによって独自の Java クライアント プロジェクトを作成し、クライアント プロジェクト内で、対象のジョブに必要なすべてのジョブ設定を指定して、実行します。

   手順については、**Java アプリケーションの作成**（37ページ）を参照してください。

設定プロパティ ファイルの使用

Big Data Quality SDKがコンピュータ上にインストールされていることを確認します。

Big Data Quality SDKジョブは、モジュール固有の JAR ファイルと XML 形式の設定ファイルを使用して実行できます。

設定プロパティのサンプルは、Big Data Quality SDK に付属しており、`<Big Data Quality bundle>\samples\configuration`にあります。

> **注：** モジュール固有の JAR ファイルの一覧については、**SDK Java API のコンポーネント**（34ページ）を参照してください。

1. Linux システムの場合は、コマンド プロンプトを起動します。
   Windows および Unix システムの場合は、Putty などの SSH クライアントを起動します。
2. *MapReduce* ジョブの場合は、コマンド `hadoop` を使用します。
   実行するジョブによって、次の操作を行います。
   1. そのモジュールの JAR ファイル名を引き渡します。
   2. ドライバ クラス名 `RunMRSampleJob`を引き渡します。
   3. 各種設定ファイルを引数リストとして渡します。各引数キーに、1つの設定プロパティ ファイルのパスが指定できます。各ファイルには、複数の設定プロパティが含まれます。

   コマンドの構文は次のとおりです。

   ```
   hadoop jar <Name of module JAR file> RunMRSampleJob [-config <Path to
   configuration file>] [-debug] [-input <Path to input configuration
   file>] [-conf <Path to MapReduce configuration file>] [-output <Path
   of output directory>]
   ```

   例えば、MapReduce MatchKeyGenerator ジョブの場合は次のようになります。

   ```
   hadoop jar amm.core.12.0.jar RunMRSampleJob -config
   /home/hadoop/matchkey/mkgConfig.xml -input
   /home/hadoop/matchkey/inputFileConfig.xml -conf
   /home/hadoop/matchkey/mapReduceConfig.xml -output
   /home/hadoop/matchkey/outputFileConfig.xml
   ```

3. *Spark* ジョブの場合は、コマンド `spark-submit` を使用します。
   実行するジョブによって、次の操作を行います。
   1. そのモジュールの JAR ファイル名を引き渡します。
   2. ドライバ クラス名 `RunSparkSampleJob`を引き渡します。
   3. 各種設定ファイルを引数リストとして渡します。各引数キーに、1つの設定プロパティ ファイルのパスが指定できます。各ファイルには、複数の設定プロパティが含まれます。

コマンドの構文は次のとおりです。

```
spark-submit --class RunSparkSampleJob <Name of module JAR file> [-config
<Path to configuration file>] [-debug] [-input <Path to input
configuration file>] [-conf <Path to Spark configuration file>] [-output
<Path of output directory>]
```

例えば、Spark MatchKeyGenerator ジョブの場合は次のようになります。

```
spark-submit --class RunSparkSampleJob amm.core.12.0.jar -config
/home/hadoop/spark/matchkey/matchKeyGeneratorConfig.xml -input
/home/hadoop/spark/matchkey/inputFileConfig.xml -output
/home/hadoop/spark/matchkey/outputFileConfig.xml
```

注： `hadoop`または `spark-submit` コマンドでサポートされる引数キーの一覧を表示するには、次のコマンドを実行します。

```
hadoop --help
```

または

```
spark-submit --help
```

Java アプリケーションの作成

Big Data Quality SDKがコンピュータ上にインストールされていることを確認します。

SDK を使用するには

1. 必要に応じて、次の方法のいずれかを使用して、SDK を使用するための Java プロジェクトを作成します。
   a) 必要なデータ品質操作を実行するための、特定の Java プロジェクトを作成する。
      この方法では、実行するデータ品質ジョブごとに、個別の Java プロジェクトを作成する必要があります。
   b) 必要なあらゆるデータ品質操作を実行するための共通の Java プロジェクトを、対応する実行時引数を使用して作成する。
      この方法では、必要なデータ品質操作に対応する実行時引数を受け入れる Java プロジェクトを、1 つのみ作成する必要があります。
2. SDK を使用するプロジェクトにBig Data Quality SDKモジュール固有の JAR ファイルをインポートします。モジュール固有の JAR ファイルの一覧については、**SDK Java API のコンポーネント**（34ページ）を参照してください。
3. 必要な Hadoop JAR ファイルをプロジェクトにインポートします。

4. 必要なデータ品質ジョブを実行するためのアプリケーションを、適切な設定を用いて作成します。
5. Maven や Ant など、任意のビルド ツールを使用してプロジェクトを構築します。
   これにより、プロジェクトの JAR ファイルが作成されます。

   例えば、`MatchKeyGeneratorClient-with-dependencies.jar`が作成されます。
6. プロジェクトの JAR ファイルを Hadoop プラットフォームに配置します。
7. Hadoop プラットフォーム上で、コマンド プロンプトを使用して、JAR ファイルを置いた場所のパスにディレクトリを移動します。
8. 次のコマンドを使用して、プロジェクトの JAR を実行します。

```
hadoop jar <name of the JAR of your client project> <fully qualified
name of the main class>
```

   例:

```
hadoop jar MatchKeyGeneratorClient-with-dependencies.jar
com.company.bdq.amm.mr.MatchKeyGeneratorJob
```

Hadoop プラットフォーム上で必要なジョブが作成され、実行されました。

Java アプリケーションは、Hadoop プラットフォーム上の指定されたパスからの入力データにアクセスし、Hadoop プラットフォーム上にジョブを作成して実行します。ジョブの出力は、Hadoop プラットフォーム上の指定された出力パスのファイルに書き出されます。

コモン API エンティティ

ConjoinedRule

目的
統合ルールの種類。複数のルールが `AND` および `OR` 演算子で結合される場合に使用されます。結合ルール (ConjoinedRule) は、シンプル ルール (SimpleRule) をコンポーネントとして含むことができます。**SimpleRule**（41ページ） を参照してください。

このクラスによって、Advanced Matching モジュールと Data Normalization モジュールのジョブのルールを定義できます。

ConsolidationCondition

目的
Advanced Matching モジュールと Data Normalization モジュールのジョブの統合ルールと対応するアクションを指定します。

ConsolidationRule

目的
レコードに対してアクションが必要かどうかの判断基準となる、統合ルールを指定します。

このクラスによって、Advanced Matching モジュールと Data Normalization モジュールのジョブの統合ルールを定義できます。

ConsolidationAction

目的
特定の統合条件のために、グループ内の他のレコードにコピーする必要のあるフィールドを指定します。

このクラスによって、Advanced Matching モジュールと Data Normalization モジュールのジョブの統合アクションを定義できます。

FilePath

目的
ジョブを実行するための入力および出力テキスト ファイルの詳細を指定します。

JobConfig<T extends ProcessType>

目的
ジョブに対して Hadoop 設定を指定するインターフェイス。

MRJobConfig

目的
MapReduce ジョブに対して Hadoop 設定を指定します。

SparkJobConfig

目的
すべての Spark ジョブに対して Hadoop 設定を指定します。

JobDetail<T extends ProcessType>

目的
ジョブの作成に必要な基本情報を保管します。

JobFactory

目的
ジョブ インスタンスの作成を指定し、作成されるジョブの詳細を指定する、ベース インターフェイス。

JobPath

目的
ジョブの入力ソースおよび出力デスティネーションの詳細を指定する親クラス。

OrcFilePath

ジョブを実行する ORC 形式ファイルの入力パスまたは出力パスを指定します。

ProcessType

目的
MapReduce や Spark など、サポートされるすべてのプロセス タイプに使用される、親マークアップ インターフェイス。

MRProcessType

目的
ジョブに対し、MapReduce プロセス タイプを指定します。

SparkProcessType

目的
ジョブに対し、Spark プロセス タイプを指定します。

ReferenceDataPath

目的
ジョブのリファレンス データのパスを指定します。

ReportManager

目的
ジョブのレポート統計を取得するためのインターフェイス。

SimpleRule

目的
統合ルールの種類。シンプル ルール (SimpleRule) は単体、または結合ルール (ConjoinedRule) のコンポーネントとして使用できます。**ConjoinedRule**（38ページ） を参照してください。

Exceptions

JobException

目的
適切なメッセージを表示して、ジョブ固有の例外を処理します。

Advanced Matching モジュールのジョブ

コモンモジュール API

AdvanceMatchDetail<T extends ProcessType>

目的
Advanced Matching モジュールのジョブの詳細を指定します。

AdvanceMatchFactory

目的
Advanced Matching モジュールのジョブのインスタンスを作成するためのシングルトン ファクトリ。

GroupbyOption<T extends ProcessType>

目的
Advanced Matching ジョブに対してグループ化を行う列を指定します。

GroupbyMROption

目的
Advanced Matching MapReduce ジョブに対してグループ化を行う列を指定します。

GroupbySparkOption

目的
Advanced Matching Spark ジョブに対してグループ化を行う列を指定します。

MatchKeySettings

目的
Match Key Generator ジョブのマッチ キーの `List` を保持します。

MatchRule

目的
Advanced Matching ジョブのマッチング ルールが作成できます。

親ノードと子ノードの階層を定義することによって、これを行います。各ノードは、マッチングされる入力フィールドの 1 つに対応します。

ChildMatchRule

目的
フィールドを特定のアルゴリズムやその他のプロパティにマッピングする、マッチ ルールの子ノードを指定します。

ParentMatchRule

目的
他の親ノードと子ノードの論理的なグループ分けである、マッチ ルールの親ノードを指定します。

特殊なシナリオ

Group-By 列が空白のレコード
Group-By の値が空白のレコードはすべて、形式に誤りがあるとみなされ、出力 HDFS フォルダへ別のファイルに出力されます。

形式に誤りがあるこれらのファイルは、次のように分類されます。

候補ファイル内の形式誤りレコード	候補ファイル内の Group-By 列が空白のレコードは、形式に誤りがあるとして破棄され、`malformedRecordsCandidate-m-<5 digit numeral>`という命名規則に従って名前が付けられたファイルに挿入されます。 例えば、`malformedRecordsCandidate-m-00000`、`malformedRecordsCandidate-m-00001` です。 これは、Interflow Match ジョブに適用されます。
サスペクト ファイル内の形式誤りレコード	サスペクト ファイル内の Group-By 列が空白のレコードは、形式に誤りがあるとして破棄され、`malformedRecordsSuspect-m-<5 digit numeral>`という命名規則に従って名前が付けられたファイルに挿入されます。

例えば、`malformedRecordsSuspect-m-00000`、`malformedRecordsSuspect-m-00001` です。

これは、Interflow Match ジョブに適用されます。

入力ファイル内の形式誤りレコード

入力ファイル内の Group-By 列が空白のレコードは、形式に誤りがあるとして破棄され、`malformedRecords-m-<5 digit numeral>`という命名規則に従って名前が付けられたファイルに挿入されます。

例えば、`malformedRecords-m-00000`、`malformedRecords-m-00001` です。

これは、Intraflow Match、Transactional Match、Best of Breed、Duplicate Synchronization、Filter のジョブに適用されます。

形式誤りレコードのカウンタ

1 回のジョブ実行における形式誤りレコードの数は、次のカウンタに保存されます。

- MALFORMED_CANDIDATE_RECORDS
- MALFORMED_SUSPECT_RECORDS
- MALFORMED_RECORDS

> **注:** これらのカウンタの値には、`AdvanceMatchFactory`インスタンスの `getCounters()` メソッドを呼び出すことによってアクセスできます。

Match Key Generator

概要

Match Key Generator ジョブでは、マッチ キーが生成できます。

> **注:** データのマッチ キーを生成するには、他のジョブを実行する前に、一度 Match Key Generator ジョブを実行する必要があります。

API エンティティ

MatchKeyGeneratorDetail

目的

Match Key Generator ジョブの詳細を指定します。

入力パラメータ

パラメータ	説明
Input File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の入力テキスト ファイルのパス。 **Record Separator** 入力ファイル内で使用されるレコード区切り文字。 **Field Separator** 入力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **Text Qualifier** 区切り記号付きファイル内のテキスト値を囲むのに使用する文字。 **Header Row Fields** 入力ファイルのヘッダー フィールドの配列。 **Skip First Row** 入力ファイル レコードの読み取り時に、先頭行をスキップするかどうかを示すフラグ。 先頭行がヘッダー行である場合は、これを `true` にする必要があります。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC File Path** Hadoop プラットフォーム上の入力 ORC 形式ファイルのパス。 *共通パラメータ:* **Field Mappings** キー値ペアのマップ。既存の列名をキーとし、対応する出力列名を値としてマッピングします。

パラメータ	説明
Output File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の出力テキスト ファイルのパス。 **Field Separator** 出力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC ファイル パス** Hadoop プラットフォーム上の出力 ORC 形式ファイルのパス。 *共通パラメータ:* **Overwrite** 出力ファイルと同じ名前のファイルが既に存在する場合に、上書きするかどうかを示すフラグ。 **Create Output Header** ヘッダー ファイルを Hadoop サーバー上に作成するかどうかを示すフラグ。
Job Configurations	ジョブ用の Hadoop 設定 MapReduce ジョブの場合、インスタンスのタイプは**MRJobConfig**（39ページ）である必要があります。Spark ジョブの場合、インスタンスのタイプは **SparkJobConfig**（40ページ）である必要があります。
Match Key Settings	マッチングの実行に必要なマッチ キーの生成に適用する、列とアルゴリズムの組み合わせ。 **注：** 少なくとも 1 つのマッチ キーを指定する必要があります。必要に応じて複数のマッチ キーを指定できます。
Job Name	ジョブの名前。

出力列

入力列に加えて、Match Key Generator ジョブの出力生成時に以下の列が追加されます。

列	説明	出力値
MatchKey	レコードを識別するために生成されるキー。	マッチ キーの生成に選択された列とアルゴリズムに基づいて生成されたキーです。 **注：** 出力で生成される、ユーザが名前を付けたマッチ キー列の数は、ジョブ設定によって異なります。

Match Key Generator MapReduce ジョブの使用

1. `AdvanceMatchFactory`のインスタンスを、その静的メソッド `getInstance()` を使用して作成します。
2. Match Key Generator ジョブの入力と出力の詳細を指定します。以下の手順に従って、`MatchKeyGeneratorDetail`を指定する`ProcessType` のインスタンスを作成することによって、これを行います。このインスタンスは、**MRProcessType**（41ページ）タイプを使用する必要があります。
   a) `MatchKeySettings`のインスタンスを作成および設定することによって、マッチングを実行するためのマッチ キー設定を指定します。詳細については、関連するコード サンプルを参照してください。
   b) `MatchKeyGeneratorDetail`のインスタンスを作成します。`JobConfig` タイプのインスタンスと、上で作成した `MatchKeySettings` インスタンスを、コンストラクタの引数として渡します。
   `JobConfig`パラメータは、**MRJobConfig**（39ページ） タイプのインスタンスである必要があります。
   c) `inputPath`インスタンスの `MatchKeyGeneratorDetail` フィールドを使用して、入力ファイルの詳細を設定します。
   テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。
   d) `outputPath`インスタンスの `MatchKeyGeneratorDetail` フィールドを使用して、出力ファイルの詳細を設定します。
   テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

e) jobNameインスタンスの MatchKeyGeneratorDetail フィールドを使用して、ジョブの名前を設定します。

3. 先ほど作成したAdvanceMatchFactoryのインスタンスを使用してそのメソッドcreateJob()を呼び出し、MapReduce ジョブを作成します。ここで、上の MatchKeyGeneratorDetail のインスタンスを引数として渡します。

   createJob()メソッドはジョブを作成し、List インスタンスの ControlledJob を返します。

4. JobControlのインスタンスを使用して、作成したジョブを実行します。

Match Key Generator Spark ジョブの使用

1. AdvanceMatchFactoryのインスタンスを、その静的メソッド getInstance() を使用して作成します。
2. Match Key Generator ジョブの入力と出力の詳細を指定します。以下の手順に従って、MatchKeyGeneratorDetail を指定する ProcessType のインスタンスを作成することによって、これを行います。このインスタンスは、**SparkProcessType**（41ページ） タイプを使用する必要があります。

   a) MatchKeySettingsのインスタンスを作成および設定することによって、マッチングを実行するためのマッチ キー設定を指定します。詳細については、関連するコード サンプルを参照してください。

   b) MatchKeyGeneratorDetail のインスタンスを作成します。JobConfig タイプのインスタンスと、上で作成した MatchKeySettings インスタンスを、コンストラクタの引数として渡します。

   JobConfig パラメータは、**SparkJobConfig**（40ページ） タイプのインスタンスである必要があります。

   c) inputPathインスタンスの MatchKeyGeneratorDetail フィールドを使用して、入力ファイルの詳細を設定します。

   テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

   d) outputPathインスタンスの MatchKeyGeneratorDetail フィールドを使用して、出力ファイルの詳細を設定します。

   テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

e) `jobName`インスタンスの `MatchKeyGeneratorDetail` フィールドを使用して、ジョブの名前を設定します。

3. Spark ジョブを作成して実行するには、先ほど作成した `AdvanceMatchFactory` のインスタンスを使用してそのメソッド `runSparkJob()` を呼び出します。ここで、上の `MatchKeyGeneratorDetail` のインスタンスを引数として渡します。

   `runSparkJob()` メソッドはジョブを実行し、ジョブのレポート カウンタの `Map` を返します。

Interflow Match

概要

Interflow ジョブでは、マッチ キーを生成し、マッチ キーを使用してレコードをグループ化し、異なるデータ ソースからのレコードに対する Interflow マッチングを実行できます。

API エンティティ

InterMatchDetail

目的
Interflow Match ジョブの詳細を指定します。

InterMatchComparisonOption

目的
Interflow Match ジョブを定義する際の比較オプション (サスペクト レコードを、すべての候補レコードと比較する必要があるか、または選択された候補レコードと比較するか) を指定します。

入力パラメータ

パラメータ	説明
Group-By Option	*MapReduce* ジョブの場合は、次の引数を渡します。 **GroupBy Column** レコードのグループ化に使用する列の名前。 **Number of Reducer Tasks** レコードのグループ化に必要なリデューサータスクの数。 *Spark* ジョブの場合、Group-Byオプションを作成するため次の引数を渡します。 **GroupBy Column** レコードのグループ化に使用する列の名前。
Match Rule	親ルールと子ルールを、`MatchRule`オブジェクトの作成に必要な数だけ定義します。 詳細については、**MatchRule**（43ページ）を参照してください。

パラメータ	説明
Candidate File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の候補テキスト ファイルのパス。 **Record Separator** 候補ファイル内で使用されるレコード区切り文字。 **Field Separator** 候補ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **Text Qualifier** 区切り記号付きファイル内のテキスト値を囲むのに使用する文字。 **Header Row Fields** 候補ファイルのヘッダー フィールドの配列。 **Skip First Row** サスペクト ファイル レコードの読み取り時に、先頭行をスキップするかどうかを示すフラグ。 先頭行がヘッダー行である場合は、これを `true` にする必要があります。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC File Path** Hadoop プラットフォーム上の入力 ORC 形式ファイルのパス。 **重要：** サスペクト ファイルと候補ファイルは、同じファイル形式である必要があります。両方ともテキスト ファイルまたは ORC 形式ファイルでなければなりません。 *共通パラメータ:* **Field Mappings** キー値ペアのマップ。既存の列名をキーとし、対応する出力列名を値としてマッピングします。

パラメータ	説明
Suspect File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上のサスペクト テキスト ファイルのパス。 **Record Separator** サスペクト ファイル内で使用されるレコード区切り文字。 **Field Separator** サスペクト ファイルで、レコード内の連続する2つのフィールドの間に使用される区切り文字。 **Text Qualifier** 区切り記号付きファイル内のテキスト値を囲むのに使用する文字。 **Header Row Fields** サスペクト ファイルのヘッダーフィールドの配列。 **Skip First Row** サスペクト ファイル レコードの読み取り時に、先頭行をスキップするかどうかを示すフラグ。 先頭行がヘッダー行である場合は、これを`true`にする必要があります。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC File Path** Hadoop プラットフォーム上の入力 ORC 形式ファイルのパス。 *共通パラメータ:* **Field Mappings** キー値ペアのマップ。既存の列名をキーとし、対応する出力列名を値としてマッピングします。

パラメータ	説明
Output File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の出力テキスト ファイルのパス。 **Field Separator** 出力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC ファイル パス** Hadoop プラットフォーム上の出力 ORC 形式ファイルのパス。 *共通パラメータ:* **Overwrite** 出力ファイルと同じ名前のファイルが既に存在する場合に、上書きするかどうかを示すフラグ。 **Create Output Header** ヘッダー ファイルを Hadoop サーバー上に作成するかどうかを示すフラグ。
Job Configurations	ジョブ用の Hadoop 設定 MapReduce ジョブの場合、インスタンスのタイプは**MRJobConfig**（39ページ）である必要があります。Spark ジョブの場合、インスタンスのタイプは**SparkJobConfig**（40ページ）である必要があります。
Match Key Settings	マッチングの実行に必要なマッチ キーの生成に適用する、列とアルゴリズムの組み合わせ。 **注：** マッチ キーを 1 つだけ指定します。 **重要：** マッチ キー設定は、マッチングを実行する前にマッチ キーを生成する場合のみ、設定します。
Job Name	ジョブの名前。
Express Match Column	レコードの Express マッチに使用する列名。

パラメータ	説明
Setting Collection Number Zero to Unique Records	ユニーク レコードのコレクション番号を 0 (ゼロ) に設定する場合は、これを `true` にします。
Comparison Option	次の 2 つのオプションのいずれかを選択できます。 • Compare the Suspect record to all Candidate records: ユニーク レコードを出力に返す必要があるかどうかを指定します。 • Compare the Suspect record to the selected Candidate record only: 検索して返す重複レコードの最大数を指定します。
Compress Output	出力を圧縮するかどうかを示すフラグ。 出力を圧縮する場合は `true` を設定します。

出力列

入力列に加えて、Interflow Match ジョブの出力生成時に以下の列が追加されます。

列	説明	出力値
Collection Number	重複レコードのコレクションを識別します。	とり得る値は、`0-0-1`、`0-0-2`、などです。
Express Match Identified	マッチの取得に Express マッチ キーが使われたかどうかを示します。	1. Express マッチ キーを使用して重複する候補レコードがマッチした場合は、出力値は `Y` になります。 2. 重複する候補レコードがマッチしたが、Express マッチ キーは使用されなかった場合は、出力値は空白になります。 3. Express マッチ キーを使用してユニークな候補レコードがマッチした場合は、出力値は `N`.になります。 4. Express マッチ キーを使用してサスペクト レコードがマッチした場合は、出力値は空白になります。
Interflow Source Type	入力レコードがサスペクト レコードであるか、候補レコードであるかを示します。	値は、サスペクト レコードの場合は `S`、候補レコードの場合は `C` になります。

列	説明	出力値
Match Record Type	コレクションに含まれるマッチ レコードのタイプを識別します。	とり得る値は、S (サスペクト レコード)、D (重複レコード)、および U (ユニーク レコード) です。
Match Score	2 つのレコードの全体的なスコアを識別します。	とり得る値は、重複レコードとユニーク レコードの場合は 0 (ゼロ) ～ 100 で、0 は精度の低いマッチ、100 は非常に精度の高いマッチを意味します。 **注：** サスペクト レコードの場合、この値は 0 です。

Interflow Match MapReduce ジョブの使用

1. `AdvanceMatchFactory`のインスタンスを、その静的メソッド `getInstance()` を使用して作成します。
2. Interflow Match ジョブの入力と出力の詳細を指定します。以下の手順に従って、`InterMatchDetail`を指定する`ProcessType` のインスタンスを作成することによって、これを行います。このインスタンスは、**MRProcessType**（41ページ）タイプを使用する必要があります。
   a) `GroupbyOption`のインスタンスを作成することによって、レコードのグループ化に使用する列を指定します。
   GroupbyMROption（42ページ）のインスタンスを使用して、必要な Group-By 列とリデューサー数を指定します。
   b) `MatchRule`のインスタンスを作成することによって、ジョブのマッチング ルールを生成します。
   c) `InterMatchDetail`のインスタンスを作成します。`JobConfig` タイプのインスタンスと、上で作成した `GroupbyOption` インスタンスおよび `MatchRule` インスタンスを、コンストラクタの引数として渡します。
   `JobConfig`パラメータは、**MRJobConfig**（39ページ） タイプのインスタンスである必要があります。
   d) `candidateFilePath`インスタンスの `InterMatchDetail` フィールドを使用して、候補ファイルの詳細を設定します。
   テキスト候補ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な候補ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 候補ファイルの場合、ORC 候補ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

e) `suspectFilePath`インスタンスの `InterMatchDetail` フィールドを使用して、サスペクト ファイルの詳細を設定します。

テキスト サスペクト ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細なサスペクト ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC サスペクト ファイルの場合、ORC サスペクト ファイルのパスを引数に指定して `OrcFilePath` のインスタンスを作成します。

重要： サスペクト ファイルと候補ファイルは、同じファイル形式である必要があります。両方ともテキスト ファイルまたは ORC 形式ファイルでなければなりません。

f) `outputPath`インスタンスの `InterMatchDetail` フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

g) `jobName`インスタンスの `InterMatchDetail` フィールドを使用して、ジョブの名前を設定します。

h) 必要に応じて、`expressMatchColumn`インスタンスの `InterMatchDetail` フィールドを使用して、Express マッチ列を設定します。

i) ユニーク レコードにコレクション番号 0 (ゼロ) を割り当てる場合は、`collectionNumberZerotoUniqueRecords`インスタンスの `InterMatchDetail` フラグに `true` を設定します。デフォルトは `true` です。

ユニーク レコードにコレクション番号 0 を割り当てたくない場合は、このフラグに `false` を設定します。

j) `comparisonOption`インスタンスの`InterMatchDetail` フィールドを使用して、比較オプションを設定します。このフィールドで、クラス **InterMatchComparisonOption**（49ページ）を使用して必要な値を設定することにより、以下の 2 つのオプションのいずれかを選択します。

- **[Compare the Suspect record to all Candidate records]**: ユニーク レコードを出力に返す必要があるかどうかを指定します。
- **[Compare the Suspect record to the selected Candidate record only]**: 検索して返す重複レコードの最大数を指定します。

k) `compressOutput`インスタンスの `InterMatchDetail` フラグに `true` を設定して、ジョブの出力を圧縮します。

l) 入力データにマッチ キーがない場合は、マッチ キー設定を指定して、Interflow Match ジョブを実行する前にまず、Match Key Generator ジョブを実行してマッチ キーを生成する必要があります。

入力データのマッチ キーを生成するには、MatchKeySettingsのインスタンスを作成および設定することによってマッチ キー設定を指定し、Interflow マッチングを実行する前にマッチ キーを生成します。matchKeySettingsインスタンスの InterMatchDetail フィールドを使用して、このインスタンスを設定します。

注： マッチ キー設定方法については、コード サンプルを参照してください。

3. 先ほど作成したAdvanceMatchFactoryのインスタンスを使用してそのメソッドcreateJob()を呼び出し、MapReduce ジョブを作成します。ここで、上の InterMatchDetailのインスタンスを引数として渡します。

   createJob()メソッドはジョブを作成し、List インスタンスの ControlledJob を返します。

4. JobControlのインスタンスを使用して、作成したジョブを実行します。
5. MapReduce ジョブの正常実行後にレポート カウンタを表示するには、先ほど作成したAdvanceMatchFactoryのインスタンスを使用して、そのメソッド getCounters() を呼び出します。作成したジョブを引数として渡します。

Interflow Match Spark ジョブの使用

1. AdvanceMatchFactoryのインスタンスを、その静的メソッド getInstance() を使用して作成します。
2. Interflow Match ジョブの入力と出力の詳細を指定します。以下の手順に従って、ProcessType を指定する InterMatchDetail のインスタンスを作成することによって、これを行います。このインスタンスは、**SparkProcessType**（41ページ） タイプを使用する必要があります。
   a) GroupbyOption のインスタンスを作成することによって、レコードのグループ化に使用する列を指定します。

      GroupbySparkOption（42ページ） のインスタンスを使用して、Group-By 列を指定します。

   b) MatchRuleのインスタンスを作成することによって、ジョブのマッチング ルールを生成します。
   c) InterMatchDetail のインスタンスを作成します。JobConfig タイプのインスタンスと、上で作成した GroupbyOption インスタンスおよび MatchRule インスタンスを、コンストラクタの引数として渡します。

      JobConfig パラメータは、**SparkJobConfig**（40ページ） タイプのインスタンスである必要があります。

   d) candidateFilePathインスタンスの InterMatchDetail フィールドを使用して、候補ファイルの詳細を設定します。

テキスト候補ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な候補ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 候補ファイルの場合、ORC 候補ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

e) `suspectFilePath`インスタンスの `InterMatchDetail` フィールドを使用して、サスペクト ファイルの詳細を設定します。

テキスト サスペクト ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細なサスペクト ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC サスペクト ファイルの場合、ORC サスペクト ファイルのパスを引数に指定して `OrcFilePath` のインスタンスを作成します。

重要： サスペクト ファイルと候補ファイルは、同じファイル形式である必要があります。両方ともテキスト ファイルまたは ORC 形式ファイルでなければなりません。

f) `outputPath`インスタンスの `InterMatchDetail` フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

g) `jobName`インスタンスの `InterMatchDetail` フィールドを使用して、ジョブの名前を設定します。

h) 必要に応じて、`expressMatchColumn`インスタンスの `InterMatchDetail` フィールドを使用して、Express マッチ列を設定します。

i) ユニーク レコードにコレクション番号 0 (ゼロ) を割り当てる場合は、`collectionNumberZerotoUniqueRecords`インスタンスの `InterMatchDetail` フラグに `true` を設定します。デフォルトは `true` です。

ユニーク レコードにコレクション番号 0 を割り当てたくない場合は、このフラグに `false` を設定します。

j) `comparisonOption`インスタンスの`InterMatchDetail` フィールドを使用して、比較オプションを設定します。このフィールドで、クラス **InterMatchComparisonOption**（49ページ）を使用して必要な値を設定することにより、以下の 2 つのオプションのいずれかを選択します。

- **[Compare the Suspect record to all Candidate records]**: ユニーク レコードを出力に返す必要があるかどうかを指定します。
- **[Compare the Suspect record to the selected Candidate record only]**: 検索して返す重複レコードの最大数を指定します。

k) `compressOutput`インスタンスの `InterMatchDetail` フラグに `true` を設定して、ジョブの出力を圧縮します。

l) 入力データにマッチ キーがない場合は、マッチ キー設定を指定して、Interflow Match ジョブを実行する前にまず、Match Key Generator ジョブを実行してマッチ キーを生成する必要があります。

入力データのマッチ キーを生成するには、`MatchKeySettings`のインスタンスを作成および設定することによってマッチ キー設定を指定し、Interflow マッチングを実行する前にマッチ キーを生成します。`matchKeySettings`インスタンスの `InterMatchDetail` フィールドを使用して、このインスタンスを設定します。

注： マッチ キー設定方法については、コード サンプルを参照してください。

3. Spark ジョブを作成して実行するには、先ほど作成した `AdvanceMatchFactory` のインスタンスを使用してそのメソッド `runSparkJob()` を呼び出します。ここで、上の `InterMatchDetail` のインスタンスを引数として渡します。
`runSparkJob()` メソッドはジョブを実行し、ジョブのレポート カウンタの `Map` を返します。
4. カウンタを表示することにより、ジョブに対する統計レポートを表示します。

Intraflow Match

概要

Intraflow ジョブでは、マッチ キーを生成し、マッチ キーを使用してレコードをグループ化し、同じデータ ソースからのレコードに対する Intraflow マッチングを実行できます。

API エンティティ

IntraMatchDetail

目的

Intraflow Match ジョブの詳細を指定します。

入力パラメータ

パラメータ	説明
Group-By Option	*MapReduce* ジョブの場合は、次の引数を渡します。 **GroupBy Column** レコードのグループ化に使用する列の名前。 **Number of Reducer Tasks** レコードのグループ化に必要なリデューサー タスクの数。 *Spark* ジョブの場合、Group-By オプションを作成するため次の引数を渡します。 **GroupBy Column** レコードのグループ化に使用する列の名前。
Match Rule	親ルールと子ルールを、`MatchRule`オブジェクトの作成に必要な数だけ定義します。 詳細については、**MatchRule**（43ページ）を参照してください。

パラメータ	説明
Input File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の入力テキスト ファイルのパス。 **Record Separator** 入力ファイル内で使用されるレコード区切り文字。 **Field Separator** 入力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **Text Qualifier** 区切り記号付きファイル内のテキスト値を囲むのに使用する文字。 **Header Row Fields** 入力ファイルのヘッダー フィールドの配列。 **Skip First Row** 入力ファイル レコードの読み取り時に、先頭行をスキップするかどうかを示すフラグ。 先頭行がヘッダー行である場合は、これを `true` にする必要があります。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC File Path** Hadoop プラットフォーム上の入力 ORC 形式ファイルのパス。 *共通パラメータ:* **Field Mappings** キー値ペアのマップ。既存の列名をキーとし、対応する出力列名を値としてマッピングします。

パラメータ	説明
Output File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の出力テキスト ファイルのパス。 **Field Separator** 出力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC ファイル パス** Hadoop プラットフォーム上の出力 ORC 形式ファイルのパス。 *共通パラメータ:* **Overwrite** 出力ファイルと同じ名前のファイルが既に存在する場合に、上書きするかどうかを示すフラグ。 **Create Output Header** ヘッダー ファイルを Hadoop サーバー上に作成するかどうかを示すフラグ。
Job Configurations	ジョブ用の Hadoop 設定 MapReduce ジョブの場合、インスタンスのタイプは**MRJobConfig**（39ページ）である必要があります。Spark ジョブの場合、インスタンスのタイプは**SparkJobConfig**（40ページ）である必要があります。
Job Name	ジョブの名前。
Express Match Column	レコードの Express マッチに使用する列名。
Setting Collection Number Zero to Unique Records	ユニーク レコードのコレクション番号を 0 (ゼロ) に設定する場合は、これを `true` にします。
Compress Output	出力を圧縮するかどうかを示すフラグ。 出力を圧縮する場合は `true` を設定します。

パラメータ	説明
Match Key Settings	マッチングの実行に必要なマッチ キーの生成に適用する、列とアルゴリズムの組み合わせ。 **注 :** マッチ キーを 1 つだけ指定します。 **重要 :** マッチ キー設定は、マッチングを実行する前にマッチ キーを生成する場合のみ、設定します。

出力列

入力列に加えて、Intraflow Match ジョブの出力生成時に以下の列が追加されます。

列	説明	出力値
Collection Number	重複レコードのコレクションを識別します。	とり得る値は、`0-0-1`、`0-0-2`、などです。
Express Match Identified	マッチの取得に Express マッチ キーが使われたかどうかを示します。	1. Express マッチ キーを使用して重複する候補レコードがマッチした場合は、出力値は `Y` になります。 2. 重複する候補レコードがマッチしたが、Express マッチ キーは使用されなかった場合は、出力値は空白になります。 3. Express マッチ キーを使用してユニークな候補レコードがマッチした場合は、出力値は空白になります。 4. Express マッチ キーを使用してサスペクト レコードがマッチした場合は、出力値は空白になります。
Match Record Type	コレクションに含まれるマッチ レコードのタイプを識別します。	とり得る値は、`S` (サスペクト レコード)、`D` (重複レコード)、および `U` (ユニーク レコード) です。
Match Score	2 つのレコードの全体的なスコアを識別します。	とり得る値は、重複レコードとユニーク レコードの場合は 0 (ゼロ) ～ 100 で、0 は精度の低いマッチ、100 は非常に精度の高いマッチを意味します。 **注 :** サスペクト レコードの場合、この値は 0 です。

Intraflow Match MapReduce ジョブの使用

1. AdvanceMatchFactoryのインスタンスを、その静的メソッド getInstance() を使用して作成します。
2. Intraflow Match ジョブの入力と出力の詳細を指定します。以下の手順に従って、IntraMatchDetailを指定するProcessType のインスタンスを作成することによって、これを行います。このインスタンスは、**MRProcessType**（41ページ）タイプを使用する必要があります。
   a) GroupbyOptionのインスタンスを作成することによって、レコードのグループ化に使用する列を指定します。

   GroupbyMROption（42ページ）のインスタンスを使用して、必要な Group-By 列とリデューサー数を指定します。

   b) MatchRuleのインスタンスを作成することによって、ジョブのマッチング ルールを生成します。
   c) IntraMatchDetailのインスタンスを作成します。JobConfig タイプのインスタンスと、上で作成した GroupbyOption インスタンスおよび MatchRule インスタンスを、コンストラクタの引数として渡します。

   JobConfigパラメータは、**MRJobConfig**（39ページ） タイプのインスタンスである必要があります。

   d) inputPathインスタンスの IntraMatchDetail フィールドを使用して、入力ファイルの詳細を設定します。

   テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

   e) outputPathインスタンスの IntraMatchDetail フィールドを使用して、出力ファイルの詳細を設定します。

   テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

   f) jobNameインスタンスの IntraMatchDetail フィールドを使用して、ジョブの名前を設定します。
   g) 必要に応じて、expressMatchColumnインスタンスの IntraMatchDetail フィールドを使用して、Express マッチ列を設定します。

h) ユニーク レコードにコレクション番号 0 (ゼロ) を割り当てる場合は、`collectionNumberZerotoUniqueRecords`インスタンスの `IntraMatchDetail` フラグに `true` を設定します。デフォルトは `true` です。

ユニーク レコードにコレクション番号 0 を割り当てたくない場合は、このフラグに `false` を設定します。

i) `compressOutput`インスタンスの `IntraMatchDetail` フラグに `true` を設定して、ジョブの出力を圧縮します。

j) 入力データにマッチ キーがない場合は、マッチ キー設定を指定して、Intraflow Match ジョブを実行する前にまず、Match Key Generator ジョブを実行してマッチ キーを生成する必要があります。

入力データのマッチ キーを生成するには、`MatchKeySettings`のインスタンスを作成および設定することによってマッチ キー設定を指定し、Intraflow マッチングを実行する前にマッチ キーを生成します。`matchKeySettings`インスタンスの `IntraMatchDetail` フィールドを使用して、このインスタンスを設定します。

注： マッチ キー設定方法については、コード サンプルを参照してください。

3. 先ほど作成した `AdvanceMatchFactory`のインスタンスを使用してそのメソッド`createJob()` を呼び出し、MapReduce ジョブを作成します。ここで、上の `IntraMatchDetail`のインスタンスを引数として渡します。

   `createJob()` メソッドはジョブを作成し、`List` インスタンスの `ControlledJob` を返します。

4. `JobControl`のインスタンスを使用して、作成したジョブを実行します。
5. MapReduce ジョブの正常実行後にレポート カウンタを表示するには、先ほど作成した `AdvanceMatchFactory`のインスタンスを使用して、そのメソッド `getCounters()` を呼び出します。作成したジョブを引数として渡します。

Intraflow Match Spark ジョブの使用

1. `AdvanceMatchFactory`のインスタンスを、その静的メソッド `getInstance()` を使用して作成します。
2. Intraflow Match ジョブの入力と出力の詳細を指定します。以下の手順に従って、`ProcessType` を指定する `IntraMatchDetail` のインスタンスを作成することによって、これを行います。このインスタンスは、**SparkProcessType**（41ページ） タイプを使用する必要があります。

   a) `GroupbyOption` のインスタンスを作成することによって、レコードのグループ化に使用する列を指定します。

   GroupbySparkOption（42ページ） のインスタンスを使用して、Group-By 列を指定します。

b) MatchRuleのインスタンスを作成することによって、ジョブのマッチング ルールを生成します。

c) IntraMatchDetail のインスタンスを作成します。JobConfig タイプのインスタンスと、上で作成した GroupbyOption インスタンスおよび MatchRule インスタンスを、コンストラクタの引数として渡します。

JobConfig パラメータは、**SparkJobConfig**（40ページ） タイプのインスタンスである必要があります。

d) inputPathインスタンスの IntraMatchDetail フィールドを使用して、入力ファイルの詳細を設定します。

テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

e) outputPathインスタンスの IntraMatchDetail フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

f) jobNameインスタンスの IntraMatchDetail フィールドを使用して、ジョブの名前を設定します。

g) 必要に応じて、expressMatchColumnインスタンスの IntraMatchDetail フィールドを使用して、Express マッチ列を設定します。

h) ユニーク レコードにコレクション番号 0 (ゼロ) を割り当てる場合は、collectionNumberZerotoUniqueRecordsインスタンスの IntraMatchDetail フラグに true を設定します。デフォルトは true です。

ユニーク レコードにコレクション番号 0 を割り当てたくない場合は、このフラグに false を設定します。

i) compressOutputインスタンスの IntraMatchDetail フラグに true を設定して、ジョブの出力を圧縮します。

j) 入力データにマッチ キーがない場合は、マッチ キー設定を指定して、Intraflow Match ジョブを実行する前にまず、Match Key Generator ジョブを実行してマッチ キーを生成する必要があります。

入力データのマッチ キーを生成するには、MatchKeySettingsのインスタンスを作成および設定することによってマッチ キー設定を指定し、Intraflow マッチングを実行する前にマッチ キーを生成します。matchKeySettingsインスタンスの IntraMatchDetail フィールドを使用して、このインスタンスを設定します。

注： マッチ キー設定方法については、コード サンプルを参照してください。

3. Spark ジョブを作成して実行するには、先ほど作成した `AdvanceMatchFactory` のインスタンスを使用してそのメソッド `runSparkJob()` を呼び出します。ここで、上の `IntraMatchDetail` のインスタンスを引数として渡します。

   `runSparkJob()` メソッドはジョブを実行し、ジョブのレポート カウンタの `Map` を返します。
4. カウンタを表示することにより、ジョブに対する統計レポートを表示します。

Transactional Match

概要

Transactional Match ジョブでは、サスペクト レコードを レコード グループの候補レコードに対してマッチングし、重複を特定することができます。

API エンティティ

TransactionalMatchDetail

目的

Transactional Match ジョブの詳細を指定します。

入力パラメータ

パラメータ	説明
Group-By Option	*MapReduce* ジョブの場合は、次の引数を渡します。 **GroupBy Column** レコードのグループ化に使用する列の名前。 **Number of Reducer Tasks** レコードのグループ化に必要なリデューサー タスクの数。 *Spark* ジョブの場合、Group-By オプションを作成するため次の引数を渡します。 **GroupBy Column** レコードのグループ化に使用する列の名前。

パラメータ	説明
Match Rule	親ルールと子ルールを、`MatchRule`オブジェクトの作成に必要な数だけ定義します。 詳細については、**MatchRule**（43ページ）を参照してください。
Input File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の入力テキスト ファイルのパス。 **Record Separator** 入力ファイル内で使用されるレコード区切り文字。 **Field Separator** 入力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **Text Qualifier** 区切り記号付きファイル内のテキスト値を囲むのに使用する文字。 **Header Row Fields** 入力ファイルのヘッダー フィールドの配列。 **Skip First Row** 入力ファイル レコードの読み取り時に、先頭行をスキップするかどうかを示すフラグ。 先頭行がヘッダー行である場合は、これを`true`にする必要があります。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC File Path** Hadoop プラットフォーム上の入力 ORC 形式ファイルのパス。 *共通パラメータ:* **Field Mappings** キー値ペアのマップ。既存の列名をキーとし、対応する出力列名を値としてマッピングします。

パラメータ	説明
Output File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の出力テキスト ファイルのパス。 **Field Separator** 出力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC ファイル パス** Hadoop プラットフォーム上の出力 ORC 形式ファイルのパス。 *共通パラメータ:* **Overwrite** 出力ファイルと同じ名前のファイルが既に存在する場合に、上書きするかどうかを示すフラグ。 **Create Output Header** ヘッダー ファイルを Hadoop サーバー上に作成するかどうかを示すフラグ。
Job Name	ジョブの名前。
Job Configurations	ジョブ用の Hadoop 設定 MapReduce ジョブの場合、インスタンスのタイプは**MRJobConfig**（39ページ）である必要があります。Spark ジョブの場合、インスタンスのタイプは **SparkJobConfig**（40ページ）である必要があります。
Return Unique Candidates	ユニークな候補を出力として返さなければならないかどうかを示すフラグ。
Compress Output	出力を圧縮するかどうかを示すフラグ。 出力を圧縮する場合は `true` を設定します。

パラメータ	説明
Match Key Settings	マッチングの実行に必要なマッチ キーの生成に適用する、列とアルゴリズムの組み合わせ。 **注 :** マッチ キーを 1 つだけ指定します。 **重要 :** マッチ キー設定は、マッチングを実行する前にマッチ キーを生成する場合のみ、設定します。

出力列

入力列に加えて、Transactional Match ジョブの出力生成時に以下の列が追加されます。

パラメータ	説明	出力値
Match Record Type	コレクションに含まれるマッチ レコードのタイプを識別します。	とり得る値は、S (サスペクト レコード)、D (重複レコード)、および U (ユニーク レコード) です。
Match Score	2 つのレコードの全体的なスコアを識別します。	とり得る値は、重複レコードとユニーク レコードの場合は 0 (ゼロ) ～ 100 で、0 は精度の低いマッチ、100 は非常に精度の高いマッチを意味します。 **注 :** サスペクト レコードの場合、この値は 0 です。
Has Duplicates	サスペクト レコードに重複があるかどうかを示します。	サスペクト レコードの場合、出力値は次のいずれかです。 • Y (重複が存在します) • N (重複は存在しません) 重複レコードの場合、出力値は D です。 ユニーク レコードの場合、出力値は U です。

Transactional Match MapReduce ジョブの使用

1. `AdvanceMatchFactory`のインスタンスを、その静的メソッド `getInstance()` を使用して作成します。
2. Transactional Match ジョブの入力と出力の詳細を指定します。以下の手順に従って、`TransactionalMatchDetail`を指定する`ProcessType` のインスタンスを作成することに

よって、これを行います。このインスタンスは、**MRProcessType**（41ページ）タイプを使用する必要があります。

a) GroupbyOptionのインスタンスを作成することによって、レコードのグループ化に使用する列を指定します。

GroupbyMROption（42ページ）のインスタンスを使用して、必要なGroup-By列とリデューサー数を指定します。

b) MatchRuleのインスタンスを作成することによって、ジョブのマッチング ルールを生成します。

c) TransactionalMatchDetailのインスタンスを作成します。JobConfig タイプのインスタンスと、上で作成した GroupbyOption インスタンスおよび MatchRule インスタンスを、コンストラクタの引数として渡します。

JobConfigパラメータは、**MRJobConfig**（39ページ） タイプのインスタンスである必要があります。

d) inputPathインスタンスの TransactionalMatchDetail フィールドを使用して、入力ファイルの詳細を設定します。

テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

e) outputPathインスタンスの TransactionalMatchDetail フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

f) jobNameインスタンスの TransactionalMatchDetail フィールドを使用して、ジョブの名前を設定します。

g) ユニークな候補レコードを出力に返す場合は、returnUniqueCandidatesインスタンスの TransactionalMatchDetail フラグに true を設定します。デフォルトは true です。

h) compressOutputインスタンスの TransactionalMatchDetail フラグに true を設定して、ジョブの出力を圧縮します。

i) 入力データにマッチ キーがない場合は、マッチ キー設定を指定して、Transactional Match ジョブを実行する前にまず、Match Key Generator ジョブを実行してマッチ キーを生成する必要があります。

入力データのマッチ キーを生成するには、MatchKeySettingsのインスタンスを作成および設定することによってマッチ キー設定を指定し、Transactional マッチングを実行する前

にマッチ キーを生成します。matchKeySettingsインスタンスの TransactionalMatchDetail フィールドを使用して、このインスタンスを設定します。

注： マッチ キー設定方法については、コード サンプルを参照してください。

3. 先ほど作成したAdvanceMatchFactoryのインスタンスを使用してそのメソッドcreateJob()を呼び出し、MapReduce ジョブを作成します。ここで、上の TransactionalMatchDetail のインスタンスを引数として渡します。

   createJob()メソッドはジョブを作成し、List インスタンスの ControlledJob を返します。

4. JobControlのインスタンスを使用して、作成したジョブを実行します。
5. MapReduce ジョブの正常実行後にレポート カウンタを表示するには、先ほど作成した AdvanceMatchFactoryのインスタンスを使用して、そのメソッド getCounters() を呼び出します。作成したジョブを引数として渡します。

Transactional Match Spark ジョブの使用

1. AdvanceMatchFactoryのインスタンスを、その静的メソッド getInstance() を使用して作成します。
2. Transactional Match ジョブの入力と出力の詳細を指定します。以下の手順に従って、ProcessType を指定する TransactionalMatchDetail のインスタンスを作成することによって、これを行います。このインスタンスは、**SparkProcessType**（41ページ） タイプを使用する必要があります。

   a) GroupbyOption のインスタンスを作成することによって、レコードのグループ化に使用する列を指定します。

   GroupbySparkOption（42ページ） のインスタンスを使用して、Group-By 列を指定します。

   b) MatchRuleのインスタンスを作成することによって、ジョブのマッチング ルールを生成します。

   c) TransactionalMatchDetail のインスタンスを作成します。JobConfig タイプのインスタンスと、上で作成した GroupbyOption インスタンスおよび MatchRule インスタンスを、コンストラクタの引数として渡します。

   JobConfig パラメータは、**SparkJobConfig**（40ページ） タイプのインスタンスである必要があります。

   d) inputPathインスタンスの TransactionalMatchDetail フィールドを使用して、入力ファイルの詳細を設定します。

   テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 入力ファイルの場

合、ORC 入力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

e) `outputPath`インスタンスの `TransactionalMatchDetail` フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

f) `jobName`インスタンスの `TransactionalMatchDetail` フィールドを使用して、ジョブの名前を設定します。

g) ユニークな候補レコードを出力に返す場合は、`returnUniqueCandidates`インスタンスの `TransactionalMatchDetail` フラグに `true` を設定します。デフォルトは `true` です。

h) `compressOutput`インスタンスの `TransactionalMatchDetail` フラグに `true` を設定して、ジョブの出力を圧縮します。

i) 入力データにマッチ キーがない場合は、マッチ キー設定を指定して、Transactional Match ジョブを実行する前にまず、Match Key Generator ジョブを実行してマッチ キーを生成する必要があります。

入力データのマッチ キーを生成するには、`MatchKeySettings`のインスタンスを作成および設定することによってマッチ キー設定を指定し、Transactional マッチングを実行する前にマッチ キーを生成します。`matchKeySettings`インスタンスの `TransactionalMatchDetail` フィールドを使用して、このインスタンスを設定します。

注： マッチ キー設定方法については、コード サンプルを参照してください。

3. Spark ジョブを作成して実行するには、先ほど作成した `AdvanceMatchFactory` のインスタンスを使用してそのメソッド `runSparkJob()` を呼び出します。ここで、上の `TransactionalMatchDetail` のインスタンスを引数として渡します。

   `runSparkJob()` メソッドはジョブを実行し、ジョブのレポート カウンタの `Map` を返します。

4. カウンタを表示することにより、ジョブに対する統計レポートを表示します。

Best of Breed

概要

Best of Breed ジョブは、 重複レコードのコレクションから最良のデータを選択し、それを使用して新しい統合レコードを作成することで、重複レコードを統合します 。

API エンティティ

BestOfBreedConfiguration

Best of Breed 統合ジョブを実行するための統合ルールとテンプレート ルールを指定します。

BestofBreedDetail

目的

Best of Breed 統合ジョブの詳細を指定します。

入力パラメータ

パラメータ	説明
Group-By Option	類似レコードのグループを結合することによって 1 つの Best of Breed レコードを作成する際に使用するフィールドを指定します。Best of Breed レコードは、各レコード グループに対して作成されます。 *MapReduce* ジョブの場合は、次の引数を渡します。 **GroupBy Column** レコードのグループ化に使用する列の名前。 **Number of Reducer Tasks** レコードのグループ化に必要なリデューサー タスクの数。 *Spark* ジョブの場合は、次の引数を渡します。 **GroupBy Column** レコードのグループ化に使用する列の名前。
Best of Breed Configuration	類似レコードの各コレクションに対して Best of Breed レコードを作成する際に使用する、統合ルールとテンプレート ルールを定義します。

パラメータ	説明
Input File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の入力テキスト ファイルのパス。 **Record Separator** 入力ファイル内で使用されるレコード区切り文字。 **Field Separator** 入力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **Text Qualifier** 区切り記号付きファイル内のテキスト値を囲むのに使用する文字。 **Header Row Fields** 入力ファイルのヘッダー フィールドの配列。 **Skip First Row** 入力ファイル レコードの読み取り時に、先頭行をスキップするかどうかを示すフラグ。 先頭行がヘッダー行である場合は、これを `true` にする必要があります。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC File Path** Hadoop プラットフォーム上の入力 ORC 形式ファイルのパス。 *共通パラメータ:* **Field Mappings** キー値ペアのマップ。既存の列名をキーとし、対応する出力列名を値としてマッピングします。

パラメータ	説明
Output File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の出力テキスト ファイルのパス。 **Field Separator** 出力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC ファイル パス** Hadoop プラットフォーム上の出力 ORC 形式ファイルのパス。 *共通パラメータ:* **Overwrite** 出力ファイルと同じ名前のファイルが既に存在する場合に、上書きするかどうかを示すフラグ。 **Create Output Header** ヘッダー ファイルを Hadoop サーバー上に作成するかどうかを示すフラグ。
Job Name	ジョブの名前。
Job Configurations	ジョブ用の Hadoop 設定 MapReduce ジョブの場合、インスタンスのタイプは**MRJobConfig**（39ページ）である必要があります。Spark ジョブの場合、インスタンスのタイプは **SparkJobConfig**（40ページ）である必要があります。
Compress Output	出力を圧縮するかどうかを示すフラグ。 出力を圧縮する場合は `true` を設定します。

出力列

入力列に加えて、Best of Breed ジョブの出力生成時に以下の列が追加されます。

パラメータ	説明	出力値
Collection Record Type	重複するレコードのコレクションからテンプレートと Best of Breed レコードを識別します。	テンプレート レコードが定義されている場合、とり得る値は次のとおりです。 **Primary** レコードが、コレクションの選択されたテンプレート レコードである場合。 **Secondary** レコードが、コレクションの選択されたテンプレート レコードでない場合。 **BestOfBreed** レコードが、コレクションの新しく作成された Best of Breed レコードである場合。 テンプレート レコードが定義されていない場合、とり得る値は **BestOfBreed** のみです。

注： **[Collection Record Type]** 以外の出力列は、Best of Breed 環境設定用の統合条件の作成時に定義された場合のみ表示されます。

Best of Breed MapReduce ジョブの使用

1. `AdvanceMatchFactory`のインスタンスを、その静的メソッド `getInstance()` を使用して作成します。
2. Best of Breed ジョブの入力と出力の詳細を指定します。以下の手順に従って、`BestofBreedDetail`を指定する`ProcessType` のインスタンスを作成することによって、これを行います。このインスタンスは、**MRProcessType**（41ページ）タイプを使用する必要があります。
   a) `GroupbyOption`のインスタンスを作成することによって、レコードのグループ化に使用する列を指定します。

   GroupbyMROption（42ページ）のインスタンスを使用して、必要な Group-By 列とリデューサー数を指定します。

b) BestOfBreedConfigurationのインスタンスを作成することによって、ジョブの統合ルールとテンプレート ルールを生成します。このインスタンスの中で、次の操作を行います。

1. ConsolidationConditionのインスタンスを使用して、テンプレート レコードを定義します。このインスタンスは、ConsolidationRule インスタンスで構成されます。
2. ConsolidationConditionのインスタンスを使用して統合条件を定義し、論理演算子を使用して条件を組み合わせます。

   ConsolidationConditionの各インスタンスは、ConsolidationRule インスタンスとそれに対応する ConsolidationAction インスタンスを使用して定義されます。

   注： ConsolidationRuleの各インスタンスは、SimpleRule の 1 つのインスタンスによって定義するか、または、子の SimpleRule インスタンスとネストされた ConjoinedRule インスタンスが、論理演算子で結合された階層を使用して定義できます。列挙 **JoinType**（201ページ）、および列挙 **Operation**（200ページ）を参照してください。

c) BestofBreedDetailのインスタンスを作成します。JobConfig タイプのインスタンスと、上で作成した GroupbyOption インスタンスおよび BestOfBreedConfiguration インスタンスを、コンストラクタの引数として渡します。

JobConfigパラメータは、**MRJobConfig**（39ページ） タイプのインスタンスである必要があります。

d) inputPathインスタンスの BestofBreedDetail フィールドを使用して、入力ファイルの詳細を設定します。

テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

e) outputPathインスタンスの BestofBreedDetail フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

f) jobNameインスタンスの BestofBreedDetail フィールドを使用して、ジョブの名前を設定します。

g) compressOutputインスタンスの BestofBreedDetail フラグに true を設定して、ジョブの出力を圧縮します。

3. 先ほど作成したAdvanceMatchFactoryのインスタンスを使用してそのメソッドcreateJob()を呼び出し、MapReduceジョブを作成します。ここで、上のBestofBreedDetailのインスタンスを引数として渡します。

   createJob()メソッドはジョブを作成し、Listインスタンスの ControlledJob を返します。

4. JobControlのインスタンスを使用して、作成したジョブを実行します。
5. MapReduce ジョブの正常実行後にレポート カウンタを表示するには、先ほど作成したAdvanceMatchFactoryのインスタンスを使用して、そのメソッド getCounters() を呼び出します。作成したジョブを引数として渡します。

Best of Breed Spark ジョブの使用

1. AdvanceMatchFactoryのインスタンスを、その静的メソッド getInstance() を使用して作成します。
2. Best of Breed ジョブの入力と出力の詳細を指定します。以下の手順に従って、ProcessTypeを指定するBestofBreedDetail のインスタンスを作成することによって、これを行います。このインスタンスは、**SparkProcessType**（41ページ） タイプを使用する必要があります。

   a) GroupbyOption のインスタンスを作成することによって、レコードのグループ化に使用する列を指定します。

   GroupbySparkOption（42ページ） のインスタンスを使用して、Group-By 列を指定します。

   b) BestOfBreedConfigurationのインスタンスを作成することによって、ジョブの統合ルールとテンプレート ルールを生成します。このインスタンスの中で、次の操作を行います。

      1. ConsolidationConditionのインスタンスを使用して、テンプレート レコードを定義します。このインスタンスは、ConsolidationRule インスタンスで構成されます。
      2. ConsolidationConditionのインスタンスを使用して統合条件を定義し、論理演算子を使用して条件を組み合わせます。

         ConsolidationConditionの各インスタンスは、ConsolidationRule インスタンスとそれに対応する ConsolidationAction インスタンスを使用して定義されます。

         注： ConsolidationRuleの各インスタンスは、SimpleRule の 1 つのインスタンスによって定義するか、または、子の SimpleRule インスタンスとネストされたConjoinedRule インスタンスが、論理演算子で結合された階層を使用して定義できます。**列挙 JoinType**（201ページ）、および**列挙 Operation**（200ページ）を参照してください。

c) `BestofBreedDetail` のインスタンスを作成します。`JobConfig` タイプのインスタンスと、上で作成した `GroupbyOption` インスタンスおよび `BestOfBreedConfiguration` インスタンスを、コンストラクタの引数として渡します。

`JobConfig` パラメータは、**SparkJobConfig**（40ページ） タイプのインスタンスである必要があります。

d) `inputPath`インスタンスの `BestofBreedDetail` フィールドを使用して、入力ファイルの詳細を設定します。

テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

e) `outputPath`インスタンスの `BestofBreedDetail` フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

f) `jobName`インスタンスの `BestofBreedDetail` フィールドを使用して、ジョブの名前を設定します。

g) `compressOutput`インスタンスの `BestofBreedDetail` フラグに `true` を設定して、ジョブの出力を圧縮します。

3. Spark ジョブを作成して実行するには、先ほど作成した `AdvanceMatchFactory` のインスタンスを使用してそのメソッド `runSparkJob()` を呼び出します。ここで、上の `BestofBreedDetail` のインスタンスを引数として渡します。

   `runSparkJob()` メソッドはジョブを実行し、ジョブのレポート カウンタの `Map` を返します。

4. カウンタを表示することにより、ジョブに対する統計レポートを表示します。

Duplicate Synchronization

概要

Duplicate Synchronization ジョブでは、 レコードのコレクションからフィールドを指定して、そのフィールドをコレクション内のすべてのレコードの対応するフィールドにコピーできます。

API エンティティ

DuplicateSynchronizationConfiguration

Duplicate Synchronization 統合ジョブを実行するための統合ルールを指定します。

DuplicateSyncDetail

目的

Duplicate Synchronization 統合ジョブの詳細を指定します。

入力パラメータ

パラメータ	説明
Group-By Option	同期するレコードのグループを作成するのに使用するフィールドを指定します。 *MapReduce* ジョブの場合は、次の引数を渡します。 **GroupBy Column** レコードのグループ化に使用する列の名前。 **Number of Reducer Tasks** レコードのグループ化に必要なリデューサー タスクの数。 *Spark* ジョブの場合、Group-By オプションを作成するため次の引数を渡します。 **GroupBy Column** レコードのグループ化に使用する列の名前。 **注 :** 入力にグループがない場合は、このパラメータに null を設定します。その場合は、データ全体が 1 つのグループであるとみなされます。
Duplicate Synchronization Configuration	あるレコードのフィールドをコレクションの他のレコードにコピーするときに適用されるルール。

パラメータ	説明
Input File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の入力テキスト ファイルのパス。 **Record Separator** 入力ファイル内で使用されるレコード区切り文字。 **Field Separator** 入力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **Text Qualifier** 区切り記号付きファイル内のテキスト値を囲むのに使用する文字。 **Header Row Fields** 入力ファイルのヘッダー フィールドの配列。 **Skip First Row** 入力ファイル レコードの読み取り時に、先頭行をスキップするかどうかを示すフラグ。 先頭行がヘッダー行である場合は、これを `true` にする必要があります。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC File Path** Hadoop プラットフォーム上の入力 ORC 形式ファイルのパス。 *共通パラメータ:* **Field Mappings** キー値ペアのマップ。既存の列名をキーとし、対応する出力列名を値としてマッピングします。

パラメータ	説明
Output File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の出力テキスト ファイルのパス。 **Field Separator** 出力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC ファイル パス** Hadoop プラットフォーム上の出力 ORC 形式ファイルのパス。 *共通パラメータ:* **Overwrite** 出力ファイルと同じ名前のファイルが既に存在する場合に、上書きするかどうかを示すフラグ。 **Create Output Header** ヘッダー ファイルを Hadoop サーバー上に作成するかどうかを示すフラグ。
Job Name	ジョブの名前。
Compress Output	出力を圧縮するかどうかを示すフラグ。 出力を圧縮する場合は `true` を設定します。

出力列

[Duplicate Synchronization Configuration] 入力パラメータで定義された統合条件に基づき、必要に応じて、入力列以外の列が出力に追加される場合があります。

Duplicate Synchronization MapReduce ジョブの使用

1. `AdvanceMatchFactory`のインスタンスを、その静的メソッド `getInstance()` を使用して作成します。
2. Duplicate Synchronization ジョブの入力と出力の詳細を指定します。以下の手順に従って、`DuplicateSyncDetail`を指定する`ProcessType` のインスタンスを作成することによって、

これを行います。このインスタンスは、**MRProcessType**（41ページ）タイプを使用する必要があります。

a) GroupbyOptionのインスタンスを作成することによって、レコードのグループ化に使用する列を指定します。

GroupbyMROption（42ページ）のインスタンスを使用して、必要なGroup-By列とリデューサー数を指定します。

b) DuplicateSynchronizationConfigurationのインスタンスを作成することによって、ジョブの統合条件を生成します。このインスタンスの中で、ConsolidationConditionのインスタンスを使用して統合条件を定義し、論理演算子を使用して条件を組み合わせます。

ConsolidationConditionの各インスタンスは、ConsolidationRule インスタンスとそれに対応する ConsolidationAction インスタンスを使用して定義されます。

> **注：** ConsolidationRuleの各インスタンスは、SimpleRule の 1 つのインスタンスによって定義するか、または、子の SimpleRule インスタンスとネストされた ConjoinedRule インスタンスが、論理演算子で結合された階層を使用して定義できます。**列挙 JoinType**（201ページ）、および**列挙 Operation**（200ページ）を参照してください。

c) DuplicateSyncDetailのインスタンスを作成します。JobConfig タイプのインスタンスと、上で作成した GroupbyOption インスタンスおよび DuplicateSynchronizationConfiguration インスタンスを、コンストラクタの引数として渡します。

JobConfigパラメータは、**MRJobConfig**（39ページ） タイプのインスタンスである必要があります。

d) inputPathインスタンスの DuplicateSyncDetail フィールドを使用して、入力ファイルの詳細を設定します。

テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

e) outputPathインスタンスの DuplicateSyncDetail フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

f) jobNameインスタンスの DuplicateSyncDetail フィールドを使用して、ジョブの名前を設定します。

g) compressOutputインスタンスの DuplicateSyncDetail フラグに true を設定して、ジョブの出力を圧縮します。

3. 先ほど作成したAdvanceMatchFactoryのインスタンスを使用してそのメソッドcreateJob()を呼び出し、ジョブを作成します。ここで、上の DuplicateSyncDetailのインスタンスを引数として渡します。
createJob()メソッドは、List のインスタンスの ControlledJob を返します。
4. JobControlのインスタンスを使用して、作成したジョブを実行します。
5. MapReduce ジョブの正常実行後にレポート カウンタを表示するには、先ほど作成したAdvanceMatchFactoryのインスタンスを使用して、そのメソッド getCounters() を呼び出します。作成したジョブを引数として渡します。

Duplicate Synchronization Spark ジョブの使用

1. AdvanceMatchFactoryのインスタンスを、その静的メソッド getInstance() を使用して作成します。
2. Duplicate Synchronization ジョブの入力と出力の詳細を指定します。以下の手順に従って、ProcessType を指定する DuplicateSyncDetail のインスタンスを作成することによって、これを行います。このインスタンスは、**SparkProcessType**（41ページ） タイプを使用する必要があります。

   a) GroupbyOption のインスタンスを作成することによって、レコードのグループ化に使用する列を指定します。

   GroupbySparkOption（42ページ） のインスタンスを使用して、Group-By 列を指定します。

   b) DuplicateSynchronizationConfigurationのインスタンスを作成することによって、ジョブの統合条件を生成します。このインスタンスの中で、ConsolidationCondition のインスタンスを使用して統合条件を定義し、論理演算子を使用して条件を組み合わせます。

   ConsolidationConditionの各インスタンスは、ConsolidationRule インスタンスとそれに対応する ConsolidationAction インスタンスを使用して定義されます。

   > **注：** ConsolidationRuleの各インスタンスは、SimpleRule の 1 つのインスタンスによって定義するか、または、子の SimpleRule インスタンスとネストされた ConjoinedRule インスタンスが、論理演算子で結合された階層を使用して定義できます。**列挙 JoinType**（201ページ）、および**列挙 Operation**（200ページ）を参照してください。

c) `DuplicateSyncDetail` のインスタンスを作成します。`JobConfig` タイプのインスタンスと、上で作成した `GroupbyOption` インスタンスおよび `DuplicateSynchronizationConfiguration` インスタンスを、コンストラクタの引数として渡します。

`JobConfig` パラメータは、**SparkJobConfig**（40ページ） タイプのインスタンスである必要があります。

d) `inputPath`インスタンスの `DuplicateSyncDetail` フィールドを使用して、入力ファイルの詳細を設定します。

テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

e) `outputPath`インスタンスの `DuplicateSyncDetail` フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

f) `jobName`インスタンスの `DuplicateSyncDetail` フィールドを使用して、ジョブの名前を設定します。

g) `compressOutput`インスタンスの `DuplicateSyncDetail` フラグに `true` を設定して、ジョブの出力を圧縮します。

3. Spark ジョブを作成して実行するには、先ほど作成した `AdvanceMatchFactory` のインスタンスを使用してそのメソッド `runSparkJob()` を呼び出します。ここで、上の `DuplicateSyncDetail` のインスタンスを引数として渡します。

   `runSparkJob()` メソッドはジョブを実行し、ジョブのレポート カウンタの `Map` を返します。

4. カウンタを表示することにより、ジョブに対する統計レポートを表示します。

フィルタ

概要

Filter ジョブでは、指定したルールに基づいて、レコードをレコードのグループに残すか、または除外します 。

API エンティティ

FilterConfiguration

Filter 統合ジョブを実行するための統合ルールを指定します。

FilterDetail

目的

Filter 統合ジョブの詳細を指定します。

入力パラメータ

パラメータ	説明
Group-By Option	フィルタリングするレコードのグループを作成するのに使用するフィールドを指定します。Filter ジョブは、各グループからのレコードを 1 つ以上保持します。 *MapReduce* ジョブの場合は、次の引数を渡します。 **GroupBy Column** レコードのグループ化に使用する列の名前。 **Number of Reducer Tasks** レコードのグループ化に必要なリデューサー タスクの数。 *Spark* ジョブの場合、Group-By オプションを作成するため次の引数を渡します。 **GroupBy Column** レコードのグループ化に使用する列の名前。 **注 :** 入力にグループがない場合は、このパラメータに null を設定します。その場合は、データ全体が 1 つのグループであるとみなされます。

パラメータ	説明
Filter Configuration	統合条件を定義します。この条件に基づいて、ジョブは各グループからのレコードを 1 つ以上保持します。
Input File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の入力テキスト ファイルのパス。 **Record Separator** 入力ファイル内で使用されるレコード区切り文字。 **Field Separator** 入力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **Text Qualifier** 区切り記号付きファイル内のテキスト値を囲むのに使用する文字。 **Header Row Fields** 入力ファイルのヘッダー フィールドの配列。 **Skip First Row** 入力ファイル レコードの読み取り時に、先頭行をスキップするかどうかを示すフラグ。 先頭行がヘッダー行である場合は、これを `true` にする必要があります。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC File Path** Hadoop プラットフォーム上の入力 ORC 形式ファイルのパス。 *共通パラメータ:* **Field Mappings** キー値ペアのマップ。既存の列名をキーとし、対応する出力列名を値としてマッピングします。

パラメータ	説明
Output File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の出力テキスト ファイルのパス。 **Field Separator** 出力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC ファイル パス** Hadoop プラットフォーム上の出力 ORC 形式ファイルのパス。 *共通パラメータ:* **Overwrite** 出力ファイルと同じ名前のファイルが既に存在する場合に、上書きするかどうかを示すフラグ。 **Create Output Header** ヘッダー ファイルを Hadoop サーバー上に作成するかどうかを示すフラグ。
Job Name	ジョブの名前。
Compress Output	出力を圧縮するかどうかを示すフラグ。 出力を圧縮する場合は `true` を設定します。

出力列

出力列は入力列と同じです。出力に追加されるその他の列はありません。

Filter MapReduce ジョブの使用

1. `AdvanceMatchFactory`のインスタンスを、その静的メソッド `getInstance()` を使用して作成します。
2. Filter ジョブの入力と出力の詳細を指定します。以下の手順に従って、`FilterDetail`を指定する`ProcessType` のインスタンスを作成することによって、これを行います。このインスタンスは、**MRProcessType**（41ページ）タイプを使用する必要があります。

a) GroupbyOptionのインスタンスを作成することによって、レコードのグループ化に使用する列を指定します。

GroupbyMROption（42ページ）のインスタンスを使用して、必要なGroup-By列とリデューサー数を指定します。

b) FilterConfigurationのインスタンスを作成することによって、ジョブの統合ルールを生成します。このインスタンスの中で、ConsolidationConditionのインスタンスを使用して統合条件を定義し、論理演算子を使用して条件を組み合わせます。

ConsolidationConditionの各インスタンスは、ConsolidationRule インスタンスとそれに対応する ConsolidationAction インスタンスを使用して定義されます。

> **注：** ConsolidationRuleの各インスタンスは、SimpleRule の 1 つのインスタンスによって定義するか、または、子の SimpleRule インスタンスとネストされた ConjoinedRule インスタンスが、論理演算子で結合された階層を使用して定義できます。**列挙 JoinType**（201ページ）、および**列挙 Operation**（200ページ）を参照してください。

c) FilterDetailのインスタンスを作成します。JobConfig タイプのインスタンスと、上で作成した GroupbyOption インスタンスおよび FilterConfiguration インスタンスを、コンストラクタの引数として渡します。

JobConfigパラメータは、**MRJobConfig**（39ページ） タイプのインスタンスである必要があります。

d) inputPathインスタンスの FilterDetail フィールドを使用して、入力ファイルの詳細を設定します。

テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

e) outputPathインスタンスの FilterDetail フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

f) jobNameインスタンスの FilterDetail フィールドを使用して、ジョブの名前を設定します。

g) compressOutputインスタンスの FilterDetail フラグに true を設定して、ジョブの出力を圧縮します。

3. 先ほど作成したAdvanceMatchFactoryのインスタンスを使用してそのメソッドcreateJob()を呼び出し、ジョブを作成します。ここで、上の FilterDetailのインスタンスを引数として渡します。
   createJob()メソッドは、List のインスタンスの ControlledJob を返します。
4. JobControlのインスタンスを使用して、作成したジョブを実行します。
5. MapReduce ジョブの正常実行後にレポート カウンタを表示するには、先ほど作成したAdvanceMatchFactoryのインスタンスを使用して、そのメソッド getCounters() を呼び出します。作成したジョブを引数として渡します。

Filter Spark ジョブの使用

1. AdvanceMatchFactoryのインスタンスを、その静的メソッド getInstance() を使用して作成します。
2. Filter ジョブの入力と出力の詳細を指定します。以下の手順に従って、ProcessType を指定する FilterDetail のインスタンスを作成することによって、これを行います。このインスタンスは、**SparkProcessType**（41ページ） タイプを使用する必要があります。
   a) GroupbyOption のインスタンスを作成することによって、レコードのグループ化に使用する列を指定します。
   GroupbySparkOption（42ページ） のインスタンスを使用して、Group-By 列を指定します。
   b) FilterConfigurationのインスタンスを作成することによって、ジョブの統合ルールを生成します。このインスタンスの中で、ConsolidationConditionのインスタンスを使用して統合条件を定義し、論理演算子を使用して条件を組み合わせます。
   ConsolidationConditionの各インスタンスは、ConsolidationRule インスタンスとそれに対応する ConsolidationAction インスタンスを使用して定義されます。

   > **注：** ConsolidationRuleの各インスタンスは、SimpleRule の 1 つのインスタンスによって定義するか、または、子の SimpleRule インスタンスとネストされた ConjoinedRule インスタンスが、論理演算子で結合された階層を使用して定義できます。**列挙 JoinType**（201ページ）、および**列挙 Operation**（200ページ）を参照してください。

   c) FilterDetail のインスタンスを作成します。JobConfig タイプのインスタンスと、上で作成した GroupbyOption インスタンスおよび FilterConfiguration インスタンスを、コンストラクタの引数として渡します。
   JobConfig パラメータは、**SparkJobConfig**（40ページ） タイプのインスタンスである必要があります。
   d) inputPathインスタンスの FilterDetail フィールドを使用して、入力ファイルの詳細を設定します。

テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

e) `outputPath`インスタンスの `FilterDetail` フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

f) `jobName`インスタンスの `FilterDetail` フィールドを使用して、ジョブの名前を設定します。

g) `compressOutput`インスタンスの `FilterDetail` フラグに `true` を設定して、ジョブの出力を圧縮します。

3. Spark ジョブを作成して実行するには、先ほど作成した `AdvanceMatchFactory` のインスタンスを使用してそのメソッド `runSparkJob()` を呼び出します。ここで、上の `FilterDetail` のインスタンスを引数として渡します。

   `runSparkJob()` メソッドはジョブを実行し、ジョブのレポート カウンタの `Map` を返します。

4. カウンタを表示することにより、ジョブに対する統計レポートを表示します。

Data Normalization モジュールのジョブ

コモンモジュール API

DataNormalizationDetail<T extends ProcessType>

目的

Data Normalization モジュールのジョブの詳細を指定します。

DataNormalizationFactory

目的
Data Normalization モジュールのジョブのインスタンスを作成するためのシングルトン ファクトリ クラス。

Table Lookup

概要
Table Lookup ジョブは、語とその語の妥当性確認済みの形式とを照合して正規化し、 標準バージョンを適用します。

API エンティティ

AbstractTableLookupRule

目的
Table Lookup で使用するルールを指定します。

Categorize

目的
Table Lookup ジョブ用の分類 (Categorize) ルールを指定します。

Identify

目的
Table Lookup ジョブ用の特定 (Identify) ルールを指定します。

Standardize

目的
Table Lookup ジョブ用の正規化 (Standardize) ルールを指定します。

TableLookupDetail

目的
Table Lookup ジョブの詳細を指定します。

TableLookupConfiguration

目的

語を、その語の妥当性確認済みの形式と照合して正規化し、正規化したバージョンをすべてのレコードに適用します。

入力パラメータ

パラメータ	説明
Table Lookup Configuration	語を、その語の妥当性確認済みの形式と照合して正規化し、正規化したバージョンをすべてのレコードに適用します。 ルールのタイプは `Standardize`、`Categorize`、または `Identify` とすることができます。
Reference Data Path	リファレンス データ パスの詳細を指定します。
Job Configurations	ジョブ用の Hadoop 設定 MapReduce ジョブの場合、インスタンスのタイプは**MRJobConfig**（39ページ）である必要があります。Spark ジョブの場合、インスタンスのタイプは**SparkJobConfig**（40ページ）である必要があります。

パラメータ	説明
Input File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の入力テキスト ファイルのパス。 **Record Separator** 入力ファイル内で使用されるレコード区切り文字。 **Field Separator** 入力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **Text Qualifier** 区切り記号付きファイル内のテキスト値を囲むのに使用する文字。 **Header Row Fields** 入力ファイルのヘッダー フィールドの配列。 **Skip First Row** 入力ファイル レコードの読み取り時に、先頭行をスキップするかどうかを示すフラグ。 先頭行がヘッダー行である場合は、これを `true` にする必要があります。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC File Path** Hadoop プラットフォーム上の入力 ORC 形式ファイルのパス。 *共通パラメータ:* **Field Mappings** キー値ペアのマップ。既存の列名をキーとし、対応する出力列名を値としてマッピングします。

パラメータ	説明
Output File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の出力テキスト ファイルのパス。 **Field Separator** 出力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC ファイル パス** Hadoop プラットフォーム上の出力 ORC 形式ファイルのパス。 *共通パラメータ:* **Overwrite** 出力ファイルと同じ名前のファイルが既に存在する場合に、上書きするかどうかを示すフラグ。 **Create Output Header** ヘッダー ファイルを Hadoop サーバー上に作成するかどうかを示すフラグ。
Job Name	ジョブの名前。
Compress Output	出力を圧縮するかどうかを示すフラグ。 出力を圧縮する場合は `true` を設定します。

出力列

入力列に加えて、Table Lookup ジョブの出力生成時に以下の列が追加されます。

列	説明	出力値
Destination	ルール オプションが `Standardize` または `Categorize` の場合、この出力列は、入力には存在しない新しい列名が、ディスティネーション列として指定された場合に追加されます。 列名は、ユーザが入力したとおりです。 **注：** Destination 列に対し、既存のソース列を選択するか、新しい列名を入力できます。	テーブル データに対してマッチングされる、ソース列の正規化された値。
Standardization Term Identified	正規化された語が識別されたかどうかを示します。	値は `Yes` または `No` です。

Table Lookup MapReduce ジョブの使用

1. `DataNormalizationFactory`のインスタンスを、その静的メソッド `getInstance()` を使用して作成します。
2. Table Lookup ジョブの入力と出力の詳細を指定します。以下の手順に従って、`TableLookupDetail`を指定する`ProcessType` のインスタンスを作成することによって、これを行います。このインスタンスは、**MRProcessType**（41ページ）タイプを使用する必要があります。
   a) `TableLookupConfiguration`のインスタンスを作成することによって、Table Lookup ルールを設定します。このインスタンスの中で、次の操作を行います。
   `AbstractTableLookupRule`タイプのインスタンスを追加します。この`AbstractTableLookupRule`インスタンスは、必要な Table Lookup ルール カテゴリに応じて `Standardize`,`Categorize` または `Identify` のいずれかのクラスを用いて定義する必要があります。
   b) `ReferenceDataPath`のインスタンスを作成することによって、リファレンス データ パスと場所のタイプの詳細を設定します。**列挙 ReferenceDataPathLocation**（200ページ）を参照してください。
   c) `TableLookupDetail`のインスタンスを作成します。`JobConfig` タイプのインスタンスと、上で作成した `TableLookupConfiguration` と `ReferenceDataPath` のインスタンスを、コンストラクタの引数として渡します。
   `JobConfig`パラメータは、**MRJobConfig**（39ページ） タイプのインスタンスである必要があります。

d) inputPathインスタンスの TableLookupDetail フィールドを使用して、入力ファイルの詳細を設定します。

テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

e) outputPathインスタンスの TableLookupDetail フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

f) jobNameインスタンスの TableLookupDetail フィールドを使用して、ジョブの名前を設定します。

g) compressOutputインスタンスの TableLookupDetail フラグに true を設定して、ジョブの出力を圧縮します。

3. 先ほど作成した DataNormalizationFactoryのインスタンスを使用してそのメソッド createJob() を呼び出し、MapReduce ジョブを作成します。ここで、上の TableLookupDetailのインスタンスを引数として渡します。
createJob()メソッドは、List のインスタンスの ControlledJob を返します。
4. JobControlのインスタンスを使用して、作成したジョブを実行します。
5. MapReduce ジョブの正常実行後にレポート カウンタを表示するには、先ほど作成した DataNormalizationFactoryのインスタンスを使用して、そのメソッド getCounters() を呼び出します。作成したジョブを引数として渡します。

Table Lookup Spark ジョブの使用

1. DataNormalizationFactoryのインスタンスを、その静的メソッド getInstance() を使用して作成します。
2. Table Lookup ジョブの入力と出力の詳細を指定します。以下の手順に従って、ProcessType を指定する TableLookupDetail のインスタンスを作成することによって、これを行います。このインスタンスは、**SparkProcessType**（41ページ） タイプを使用する必要があります。

a) TableLookupConfigurationのインスタンスを作成することによって、Table Lookup ルールを設定します。このインスタンスの中で、次の操作を行います。

AbstractTableLookupRuleタイプのインスタンスを追加します。この AbstractTableLookupRuleインスタンスは、必要な Table Lookup ルール カテゴリに応じて Standardize,Categorize または Identify のいずれかのクラスを用いて定義する必要があります。

b) `ReferenceDataPath`のインスタンスを作成することによって、リファレンス データ パスと場所のタイプの詳細を設定します。**列挙 ReferenceDataPathLocation**（200ページ）を参照してください。

c) `TableLookupDetail` のインスタンスを作成します。`JobConfig` タイプのインスタンスと、上で作成した `TableLookupConfiguration` と `ReferenceDataPath` のインスタンスを、コンストラクタの引数として渡します。

`JobConfig` パラメータは、**SparkJobConfig**（40ページ） タイプのインスタンスである必要があります。

d) `inputPath`インスタンスの `TableLookupDetail` フィールドを使用して、入力ファイルの詳細を設定します。

テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

e) `outputPath`インスタンスの `TableLookupDetail` フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

f) `jobName`インスタンスの `TableLookupDetail` フィールドを使用して、ジョブの名前を設定します。

g) `compressOutput`インスタンスの `TableLookupDetail` フラグに `true` を設定して、ジョブの出力を圧縮します。

3. Spark ジョブを作成して実行するには、先ほど作成した `DataNormalizationFactory` のインスタンスを使用してそのメソッド `runSparkJob()` を呼び出します。ここで、上の `TableLookupDetail` のインスタンスを引数として渡します。

`runSparkJob()` メソッドはジョブを実行し、ジョブのレポート カウンタの `Map` を返します。

4. カウンタを表示することにより、ジョブに対する統計レポートを表示します。

Advanced Transformer

概要

Advanced Transformer ジョブは、テーブルまたは正規表現を使用して、一連のデータ列をスキャンして複数のフィールドに分割します。特定の語や、語の右側または左側から指定した数の単語を抽出します 。

API エンティティ

AbstractAdvancedTransformerRules

目的

Advanced Transformer ジョブのルールを指定する親クラス。

AdvancedTransformerDetail

目的

Advanced Transformer ジョブの詳細を指定します。

AdvancedTransformerConfiguration

目的

テーブルまたは正規表現を使用して、一連のデータ列をスキャンして複数のフィールドに分割します。

RegularExpressionExtraction

目的

正規表現を用いてデータを抽出するルールを指定します。

RegularExpressionGroupItem

目的

親の正規表現の一部を指定します。親の正規表現の各部分は、異なる出力フィールドに格納できます。

TableDataExtraction

目的

テーブルからデータを抽出するためのルールを定義します。

入力パラメータ

パラメータ	説明
Advanced Transformer Configuration	テーブルまたは正規表現を使用して、一連のデータ列をスキャンして複数のフィールドに分割します。 特定の語や、語の右側または左側から指定した数の単語を抽出できます。抽出データと非抽出データは、既存のフィールドまたは新しいフィールドに配置されます。 Advanced Transformer ルールは、`AdvancedTransformerConfiguration` タイプのインスタンスを用いて定義できます。このインスタンスは、`TableDataExtraction` または `RegularExpressionExtraction` のインスタンスである必要があります。
Reference Data Path	リファレンス データ パスの詳細を指定します。
Job Configurations	ジョブ用の Hadoop 設定 MapReduce ジョブの場合、インスタンスのタイプは**MRJobConfig**（39ページ）である必要があります。Spark ジョブの場合、インスタンスのタイプは **SparkJobConfig**（40ページ）である必要があります。

パラメータ	説明
Input File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の入力テキスト ファイルのパス。 **Record Separator** 入力ファイル内で使用されるレコード区切り文字。 **Field Separator** 入力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **Text Qualifier** 区切り記号付きファイル内のテキスト値を囲むのに使用する文字。 **Header Row Fields** 入力ファイルのヘッダー フィールドの配列。 **Skip First Row** 入力ファイル レコードの読み取り時に、先頭行をスキップするかどうかを示すフラグ。 先頭行がヘッダー行である場合は、これを `true` にする必要があります。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC File Path** Hadoop プラットフォーム上の入力 ORC 形式ファイルのパス。 *共通パラメータ:* **Field Mappings** キー値ペアのマップ。既存の列名をキーとし、対応する出力列名を値としてマッピングします。

パラメータ	説明
Output File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の出力テキスト ファイルのパス。 **Field Separator** 出力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC ファイル パス** Hadoop プラットフォーム上の出力 ORC 形式ファイルのパス。 *共通パラメータ:* **Overwrite** 出力ファイルと同じ名前のファイルが既に存在する場合に、上書きするかどうかを示すフラグ。 **Create Output Header** ヘッダー ファイルを Hadoop サーバー上に作成するかどうかを示すフラグ。
Job Name	ジョブの名前。

出力列

入力列に加えて、Advanced Transformer ジョブの出力生成時に以下の列が追加されます。

列	説明	出力値
Non-Extracted Data	この出力列は、入力には存在しない新しい列名が、Non-Extracted Data 列として指定された場合に追加されます。 列名は、ユーザが入力したとおりです。 **注：** Non-Extracted Data 列に対し、既存のソース列を選択するか、新しい列名を入力できます。	指定された語に基づく、各レコードの非抽出データ。

列	説明	出力値
Extracted Data	この出力列は、入力には存在しない新しい列名が、Extracted Data 列として指定された場合に追加されます。 列名は、ユーザが入力したとおりです。 **注：** Extracted Data 列に対し、既存のソース列を選択するか、新しい列名を入力できます。	指定された語に基づく、各レコードの抽出データ。
Advanced Transform Term Identified	語が識別されたかどうかを示します。	値は `Yes` または `No` です。

Advanced Transformer MapReduce ジョブの使用

1. `DataNormalizationFactory`のインスタンスを、その静的メソッド `getInstance()` を使用して作成します。
2. Advanced Transformer ジョブの入力と出力の詳細を指定します。以下の手順に従って、`AdvancedTransformerDetail`を指定する`ProcessType` のインスタンスを作成することによって、これを行います。このインスタンスは、**MRProcessType**（41ページ）タイプを使用する必要があります。
   a) `AdvancedTransformerConfiguration`のインスタンスを作成することによって、Advanced Transformer ルールを設定します。このインスタンスの中で、次の操作を行います。
   `AbstractAdvancedTransformerRules`タイプのインスタンスを追加します。この`AbstractAdvancedTransformerRules`インスタンスは、必要な Advanced Transformer ルール カテゴリに応じて `TableDataExtraction` または `RegularExpressionExtraction` のいずれかのクラスを用いて定義する必要があります。
   b) `ReferenceDataPath`のインスタンスを作成することによって、リファレンス データ パスと場所のタイプの詳細を設定します。**列挙 ReferenceDataPathLocation**（200ページ）を参照してください。
   c) `AdvancedTransformerDetail`のインスタンスを作成します。`JobConfig` タイプのインスタンスと、上で作成した `AdvancedTransformerConfiguration` と `ReferenceDataPath` のインスタンスを、コンストラクタの引数として渡します。
   `JobConfig`パラメータは、**MRJobConfig**（39ページ） タイプのインスタンスである必要があります。
   d) `inputPath`インスタンスの `AdvancedTransformerDetail` フィールドを使用して、入力ファイルの詳細を設定します。

テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

e) outputPathインスタンスの AdvancedTransformerDetail フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

f) jobNameインスタンスの AdvancedTransformerDetail フィールドを使用して、ジョブの名前を設定します。

3. 先ほど作成した DataNormalizationFactoryのインスタンスを使用してそのメソッド createJob() を呼び出し、MapReduce ジョブを作成します。ここで、上の AdvancedTransformerDetailのインスタンスを引数として渡します。createJob()メソッドは、List のインスタンスの ControlledJob を返します。
4. JobControlのインスタンスを使用して、作成したジョブを実行します。
5. MapReduce ジョブの正常実行後にレポート カウンタを表示するには、先ほど作成した DataNormalizationFactoryのインスタンスを使用して、そのメソッド getCounters() を呼び出します。作成したジョブを引数として渡します。

Advanced Transformer Spark ジョブの使用

1. DataNormalizationFactoryのインスタンスを、その静的メソッド getInstance() を使用して作成します。
2. Advanced Transformer ジョブの入力と出力の詳細を指定します。以下の手順に従って、ProcessType を指定する AdvancedTransformerDetail のインスタンスを作成することによって、これを行います。このインスタンスは、**SparkProcessType**（41ページ） タイプを使用する必要があります。

a) AdvancedTransformerConfigurationのインスタンスを作成することによって、Advanced Transformer ルールを設定します。このインスタンスの中で、次の操作を行います。

AbstractAdvancedTransformerRulesタイプのインスタンスを追加します。この AbstractAdvancedTransformerRulesインスタンスは、必要な Advanced Transformer ルール カテゴリに応じて TableDataExtraction または RegularExpressionExtractionのいずれかのクラスを用いて定義する必要があります。

b) ReferenceDataPathのインスタンスを作成することによって、リファレンス データ パスと場所のタイプの詳細を設定します。列挙 **ReferenceDataPathLocation**（200ページ）を参照してください。

c) `AdvancedTransformerDetail` のインスタンスを作成します。`JobConfig` タイプのインスタンスと、上で作成した `AdvancedTransformerConfiguration` と `ReferenceDataPath` のインスタンスを、コンストラクタの引数として渡します。

`JobConfig` パラメータは、**SparkJobConfig**（40ページ） タイプのインスタンスである必要があります。

d) `inputPath`インスタンスの `AdvancedTransformerDetail` フィールドを使用して、入力ファイルの詳細を設定します。

テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

e) `outputPath`インスタンスの `AdvancedTransformerDetail` フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

f) `jobName`インスタンスの `AdvancedTransformerDetail` フィールドを使用して、ジョブの名前を設定します。

3. Spark ジョブを作成して実行するには、先ほど作成した `DataNormalizationFactory` のインスタンスを使用してそのメソッド `runSparkJob()` を呼び出します。ここで、上の `AdvancedTransformerDetail` のインスタンスを引数として渡します。

   `runSparkJob()` メソッドはジョブを実行し、ジョブのレポート カウンタの `Map` を返します。

4. カウンタを表示することにより、ジョブに対する統計レポートを表示します。

Universal Addressing モジュールのジョブ

コモンモジュール API

UniversalAddressingDetail<T extends ProcessType>

目的
Universal Addressing モジュールのジョブの詳細を指定します。

UniversalAddressingFactory

目的
Universal Addressing モジュールのジョブのインスタンスを作成するためのシングルトン ファクトリ クラス。

Validate Address

API エンティティ

UAMAddressingDetail<T extends ProcessType>

目的
Validate Address ジョブの詳細を指定します。

UniversalAddressEngineConfiguration

目的
Validate Address ジョブの作成および実行に必要な *リファレンス データ パス*や *COBOL* ランタイム パス等の各種設定を行います。

これらは 1 回限りの設定です。

UAMAddressingFactory

目的

Validate Address ジョブのインスタンスを作成するためのシングルトン ファクトリ クラス。

このインスタンスは、レポート カウンタと CASS レポートを生成するために使われます。

UniversalAddressGeneralConfiguration

目的

Validate Address ジョブを作成および実行するために必要な JVM 構成を設定します。

UniversalAddressValidateInputConfiguration

目的

Validate Address ジョブを作成および実行するために入力の設定を構成する。これはルール設定であり、さまざまなオプションがあります。これらの設定はジョブごとに異なります。

入力パラメータ

パラメータ	説明
Universal Address エンジン設定	各種ジョブ実行設定を指定します。 1. DPV データベース パス 2. Suite Link DB パス 3. EWS データベース パス 4. RDI データベース パス 5. Lacs データベース パス 6. Reference Data Path 7. COBOL ランタイム パス 8. モジュール ディレクトリ

パラメータ	説明
Universal Address 検証入力設定	

パラメータ	説明
	入力設定を指定します。 **1.** 標準住所を出力 **2.** 住所要素を出力 **3.** 郵便データを出力 **4.** パース済み入力を出力 **5.** 出力住所ブロック **6.** 検索失敗時に出力をフォーマット **7.** 出力の大文字と小文字の区別 **8.** 郵便番号区切り文字を出力 **9.** 多国籍文字を出力 **10.** DPV を実行 **11.** RDI を実行 **12.** ESM を実行 **13.** ASM を実行 **14.** EWS を実行 **15.** LACS Link を実行 **16.** LOT を実行 **17.** CMRA との一致をマッチとみなさない **18.** 企業を抽出 **19.** 都市部を抽出 **20.** レポート 3553 を出力 **21.** レポート SERP を出力 **22.** レポート サマリを出力 **23.** CASS 詳細を出力 **24.** フィールドレベルのリターン コードを出力 **25.** 複数一致を保持 **26.** 結果の最大数 **27.** 標準住所フォーマット **28.** 標準住所 PMB 行 **29.** 都市名フォーマット **30.** 長い非正式都市フォーマット **31.** 国フォーマットを出力 **32.** 自国 **33.** 通りマッチングの精度 **34.** 企業マッチングの精度 **35.** 道順マッチングの精度 **36.** 二重住所ロジック **37.** DPV 成功ステータス条件 **38.** レポート リスト ファイル名 **39.** レポート リスト プロセッサ名 **40.** レポート リスト番号

パラメータ	説明
	41. レポート差出人の住所 **42.** レポート差出人の名前 **43.** レポート差出人住所の都市名行 **44.** 失敗時の住所行検索 **45.** ストリート名エイリアスを出力 **46.** VeriMove ブロックを出力 **47.** DPV で No Stat を調査 **48.** DPV で空家かどうかを調査 **49.** 省略形エイリアスを出力 **50.** よく使用されるエイリアスを出力 **51.** 優先する都市を出力 **52.** Suite Link を実行 **53.** Zplus 疑似キャリア R777 を抑制
Universal Address の全般的な設定	JVM 設定を指定します。 **1.** DPV ファイル タイプ **2.** DPV メモリ モデル **3.** Lacs Link メモリ モデル **4.** Suite Link メモリ モデル
Job Configurations	ジョブ用の Hadoop 設定 MapReduce ジョブの場合、インスタンスのタイプは**MRJobConfig**（39ページ）である必要があります。Spark ジョブの場合、インスタンスのタイプは**SparkJobConfig**（40ページ）である必要があります。

パラメータ	説明
Input File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の入力テキスト ファイルのパス。 **Record Separator** 入力ファイル内で使用されるレコード区切り文字。 **Field Separator** 入力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **Text Qualifier** 区切り記号付きファイル内のテキスト値を囲むのに使用する文字。 **Header Row Fields** 入力ファイルのヘッダー フィールドの配列。 **Skip First Row** 入力ファイル レコードの読み取り時に、先頭行をスキップするかどうかを示すフラグ。 先頭行がヘッダー行である場合は、これを `true` にする必要があります。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC File Path** Hadoop プラットフォーム上の入力 ORC 形式ファイルのパス。 *共通パラメータ:* **Field Mappings** キー値ペアのマップ。既存の列名をキーとし、対応する出力列名を値としてマッピングします。

パラメータ	説明
Output File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の出力テキスト ファイルのパス。 **Field Separator** 出力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC ファイル パス** Hadoop プラットフォーム上の出力 ORC 形式ファイルのパス。 *共通パラメータ:* **Overwrite** 出力ファイルと同じ名前のファイルが既に存在する場合に、上書きするかどうかを示すフラグ。 **Create Output Header** ヘッダー ファイルを Hadoop サーバー上に作成するかどうかを示すフラグ。
Job Name	ジョブの名前。
Compress Output	出力を圧縮するかどうかを示すフラグ。 出力を圧縮する場合は `true` を設定します。

パラメータ	説明
CASS レポート	CASS レポートを生成するための設定。`UAMAddressingFactory`インスタンスを使用して、多重定義メソッド `generateCASSReport()` のいずれかを呼び出します。 CASS レポートは PDF 形式で生成されます。 パラメータは次のとおりです。 **Counters** CASS レポートに含めるカウンタのマップ。 **Job Name** ジョブの名前。これは、CASS レポートのファイル名の一部になります。 **Path** 作成された CASS レポートが配置されるディレクトリ。これは、CASS レポートのオプションの入力値です。 `path`は、SDK ジョブがクラスタ環境で実行している場合はクラスタ上、クライアント コンピュータ上で実行している場合はクライアント コンピュータ上の場所である必要があります。 **注:** `path`が指定されていない場合は、現在の作業ディレクトリに新しい CASS レポートが配置されます。 **Report Type** 生成される CASS レポートのタイプ。**列挙 UAMCASSReportType**（210ページ）からの 1 つ以上の値を指定できます。

出力列

1. AdditionalInputData
2. AddressLine1
3. AddressLine2
4. AddressLine3
5. AddressLine4:
6. AddressLine5
7. City
8. Country
9. FirmName
10. PostalCode
11. PostalCode.AddOn
12. PostalCode.Base
13. StateProvince
14. USUrbanName
15. AdditionalInputData
16. ApartmentLabel
17. ApartmentLabel2

18. ApartmentNumber
19. ApartmentNumber2
20. HouseNumber
21. LeadingDirectional
22. POBox
23. PrivateMailbox
24. PrivateMailbox.Type
25. RRHC
26. StateProvince
27. StreetName
28. StreetSuffix
29. TrailingDirectional
30. USUrbanName
31. ApartmentLabel.Input
32. ApartmentNumber.Input
33. City.Input
34. Country.Input
35. FirmName.Input
36. HouseNumber.Input
37. LeadingDirectional.Input
38. POBox.Input
39. PostalCode.Input
40. PrivateMailbox.Input
41. PrivateMailbox.Type.Input
42. RRHC.Input
43. StateProvince.Input
44. StreetName.Input
45. StreetSuffix.Input
46. TrailingDirectional.Input
47. USUrbanName.Input
48. PostalBarCode
49. USAltAddr
50. USBCCheckDigit
51. USCarrierRouteCode
52. USCongressionalDistrict
53. USCountyName
54. USFinanceNumber
55. USFIPSCountyNumber
56. USLACS
57. USLastLineNumber
58. AddressFormat

59. Confidence
60. CouldNotValidate
61. CountryLevel
62. MatchScore
63. MultimatchCount
64. MultipleMatches
65. ProcessedBy
66. RecordType
67. RecordType.Default
68. Status
69. Status.Code
70. Status.Description
71. AddressRecord.Result
72. ApartmentLabel.Result
73. ApartmentNumber.Result
74. City.Result
75. Country.Result
76. FirmName.Result
77. HouseNumber.Result
78. LeadingDirectional.Result
79. POBox.Result
80. PostalCode.Result
81. PostalCodeCity.Result
82. PostalCode.Source
83. PostalCode.Type
84. RRHC.Result
85. RRHC.Type
86. StateProvince.Result
87. Street.Result
88. StreetName.AbbreviatedAlias.Result
89. StreetName.Alias.Type
90. StreetName.PreferredAlias.Result
91. StreetName.Result
92. StreetSuffix.Result
93. TrailingDirectional.Result
94. USUrbanName.Result
95. USLOTCode
96. USLOTHex
97. USLOTSequence
98. USLACS.ReturnCode
99. RDI

100 DPV
101 CMRA
102 DPVFootnote
103 DPVVacant
104 DPVNoStat
105 SuiteLinkReturnCode
106 SuiteLinkMatchCode
107 SuiteLinkFidelity
108 VeriMoveDataBlock

注： フィールドの説明については、Spectrum™ Technology Platformの『*Addressing ガイド*』のトピック「*Validate Address*」を参照してください。

Validate Address MapReduce ジョブの使用

重要： 最初の Validate Address ジョブを作成および実行する前に、Acushare サービスが実行されていることを確認します。手順については、**Acushare サービスの実行**（12ページ）を参照してください。

1. `UAMAddressingFactory`のインスタンスを、その静的メソッド `getInstance()` を使用して作成します。
2. `UAMAddressingDetail`を指定する`ProcessType` のインスタンスを作成して、Validate Address ジョブの入力と出力の詳細を指定します。このインスタンスは、**MRProcessType**（41ページ）タイプを使用する必要があります。これを行うには、次の手順に従います。
   a) このジョブの入力設定を行うには、`UniversalAddressValidateInputConfiguration` のインスタンスを作成します。

   列挙体 **列挙 PreferredCity**（209ページ）、**列挙 CasingType**（207ページ）、**列挙 CityNameFormat**（207ページ）、**列挙 OutputCountryFormat**（207ページ）、**列挙 StandardAddressFormat**（208ページ）、**列挙 StandardAddressPMBLine**（209ページ）、**列挙 StreetMatchingStrictness**（208ページ）、**列挙 FirmMatchingStrictness**（208ページ）、**列挙 DirectionalMatchingStrictness**（208ページ）、**列挙 DualAddressLogic**（208ページ）と該当する場合は **列挙 DPVSuccessStatusCondition**（210ページ）を使用して、さまざまな必須フィールドの値を設定します。

   重要： Validate Address を CASS 認定™ モードで実行するには、このインスタンスのフィールド `outputReport3553`、`outputCASSDetail`、および `outputReportSummary` を `true` に設定します。CASS レポートにはジョブを CASS 認定™ モードで実行した場合にのみ有効なコンテンツが含められます。それ以外の場合は、空白のレポート PDF が生成されます。

   b) `LocalReferenceDataPath`のインスタンスを作成することによって、*リファレンス データ パス*の詳細を設定します。

c) 各種ジョブ実行設定を行うために、上で作成した UAMUSAddressingEngineConfigurationインスタンスと、*COBOL* ランタイム パスおよびモジュール ディレクトリ パス (String 値) を引数としてコンストラクタに渡して、LocalReferenceDataPath のインスタンスを作成します。

UAMUSAddressingEngineConfigurationインスタンスを作成した後、その各種必須フィールドの値を設定します。

d) JVM 設定を構成するには、UniversalAddressGeneralConfigurationのインスタンスを作成します。

列挙体 **列挙 DPVFileType**（209ページ）、**列挙 DPVMemoryModel**（209ページ）、**列挙 LacsLinkMemoryModel**（209ページ）、および **列挙 SuiteLinkMemoryModel**（210ページ）を使用します。

e) UAMAddressingDetailのインスタンスを作成します。その際、JobConfig タイプのインスタンスと、上で作成した UAMUSAddressingEngineConfiguration、UniversalAddressGeneralConfiguration、および UniversalAddressValidateInputConfiguration のインスタンスを引数としてコンストラクタに渡します。

JobConfigパラメータは、**MRJobConfig**（39ページ） タイプのインスタンスである必要があります。

1. inputPathインスタンスの UAMAddressingDetail フィールドを使用して、入力ファイルの詳細を設定します。

   テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

2. outputPathインスタンスの UAMAddressingDetail フィールドを使用して、出力ファイルの詳細を設定します。

   テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

3. jobNameインスタンスの UAMAddressingDetail フィールドを使用して、ジョブの名前を設定します。
4. compressOutputインスタンスの UAMAddressingDetail フラグに true を設定して、ジョブの出力を圧縮します。

3. 先ほど作成したUAMAddressingFactoryのインスタンスを使用してそのメソッドcreateJob()を呼び出し、MapReduce ジョブを作成します。ここで、上のUAMAddressingDetailのインスタンスを引数として渡します。
   createJob()メソッドは、List のインスタンスの ControlledJob を返します。
4. JobControlのインスタンスを使用して、作成したジョブを実行します。
5. ジョブの正常実行後にレポート カウンタを表示するには、先ほど作成したUAMAddressingFactoryのインスタンスを使用して、そのメソッド getCounters() を呼び出します。作成したジョブを引数として渡します。
   カウンタの Mapが返されます。
6. ジョブの正常実行後に CASS レポートを生成するには、先ほど作成したUAMAddressingFactoryのインスタンスを使用してgenerateCASSReport() メソッドを呼び出します。多重定義されている generateCASSReport()メソッドのどのバージョンを呼び出しても構いません。

   使用される generateCASSReport()メソッド シグネチャによって、1 つ前の手順で得られたレポート カウンタの Map、jobName、生成された CASS レポートを格納する path、作成する reportType を引数として渡します。

   pathは、SDK ジョブがクラスタ環境で実行している場合はクラスタ上、クライアント コンピュータ上で実行している場合はクライアント コンピュータ上の場所である必要があります。

   > **注：** pathが指定されていない場合は、現在の作業ディレクトリに新しい CASS レポートが配置されます。

   reportTypeパラメータの値は、**列挙 UAMCASSReportType**（210ページ）に記載された値でなければなりません。1 つ以上のレポート タイプをこのパラメータに指定できます。

Validate Address Spark ジョブの使用

重要： 最初の Validate Address ジョブを作成および実行する前に、Acushare サービスが実行されていることを確認します。手順については、**Acushare サービスの実行**（12ページ）を参照してください。

1. UAMAddressingFactoryのインスタンスを、その静的メソッド getInstance() を使用して作成します。
2. UAMAddressingDetailを指定するProcessType のインスタンスを作成して、Validate Address ジョブの入力と出力の詳細を指定します。このインスタンスは、**SparkProcessType**（41ページ）タイプを使用する必要があります。これを行うには、次の手順に従います。
   a) このジョブの入力設定を行うには、UniversalAddressValidateInputConfiguration のインスタンスを作成します。

列挙体 **列挙 PreferredCity**（209ページ）、**列挙 CasingType**（207ページ）、**列挙 CityNameFormat**（207ページ）、**列挙 OutputCountryFormat**（207ページ）、**列挙 StandardAddressFormat**（208ページ）、**列挙 StandardAddressPMBLine**（209ページ）、**列挙 StreetMatchingStrictness**（208ページ）、**列挙 FirmMatchingStrictness**（208ページ）、**列挙 DirectionalMatchingStrictness**（208ページ）、**列挙 DualAddressLogic**（208ページ）と該当する場合は **列挙 DPVSuccessStatusCondition**（210ページ）を使用して、さまざまな必須フィールドの値を設定します。

重要： Validate Address を CASS 認定™ モードで実行するには、このインスタンスのフィールド `outputReport3553`、`outputCASSDetail`、および `outputReportSummary` を `true` に設定します。CASS レポートにはジョブを CASS 認定™ モードで実行した場合にのみ有効なコンテンツが含められます。それ以外の場合は、空白のレポート PDF が生成されます。

b) `LocalReferenceDataPath`のインスタンスを作成することによって、*リファレンス データ パス*の詳細を設定します。

c) 各種ジョブ実行設定を行うために、上で作成した `UAMUSAddressingEngineConfiguration`インスタンスと、*COBOL ランタイム パス*および*モジュール ディレクトリ パス* (`String` 値) を引数としてコンストラクタに渡して、`LocalReferenceDataPath` のインスタンスを作成します。

`UAMUSAddressingEngineConfiguration`インスタンスを作成した後、その各種必須フィールドの値を設定します。

d) JVM 設定を構成するには、`UniversalAddressGeneralConfiguration`のインスタンスを作成します。

列挙体 **列挙 DPVFileType**（209ページ）、**列挙 DPVMemoryModel**（209ページ）、**列挙 LacsLinkMemoryModel**（209ページ）、および **列挙 SuiteLinkMemoryModel**（210ページ）を使用します。

e) `UAMAddressingDetail`のインスタンスを作成します。その際、`JobConfig` タイプのインスタンスと、上で作成した `UAMUSAddressingEngineConfiguration`、`UniversalAddressGeneralConfiguration`、および `UniversalAddressValidateInputConfiguration` のインスタンスを引数としてコンストラクタに渡します。

`JobConfig`パラメータは、**SparkJobConfig**（40ページ）タイプのインスタンスである必要があります。

1. `inputPath`インスタンスの `UAMAddressingDetail` フィールドを使用して、入力ファイルの詳細を設定します。

   テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 入力ファイ

ルの場合、ORC 入力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

2. `outputPath`インスタンスの `UAMAddressingDetail` フィールドを使用して、出力ファイルの詳細を設定します。

   テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

3. `jobName`インスタンスの `UAMAddressingDetail` フィールドを使用して、ジョブの名前を設定します。
4. `compressOutput`インスタンスの `UAMAddressingDetail` フラグに `true` を設定して、ジョブの出力を圧縮します。

3. Spark ジョブを作成して実行するには、先ほど作成した `UAMAddressingFactory`のインスタンスを使用してそのメソッド`runSparkJob()` を呼び出します。ここで、上の `UAMAddressingDetail`のインスタンスを引数として渡します。

   `runSparkJob()`メソッドはジョブを実行し、ジョブのレポート カウンタの `Map` を返します。

4. ジョブの正常実行後にレポート カウンタを表示するには、先ほど作成した `UAMAddressingFactory`のインスタンスを使用して、そのメソッド `getCounters()` を呼び出します。作成したジョブを引数として渡します。
   カウンタの `Map`が返されます。
5. ジョブの正常実行後に CASS レポートを生成するには、先ほど作成した `UAMAddressingFactory`のインスタンスを使用して`generateCASSReport()` メソッドを呼び出します。多重定義されている `generateCASSReport()`メソッドのどのバージョンを呼び出しても構いません。

   使用される `generateCASSReport()`メソッド シグネチャによって、1 つ前の手順で得られたレポート カウンタの `Map`、`jobName`、生成された CASS レポートを格納する `path`、作成する `reportType` を引数として渡します。

   `path`は、SDK ジョブがクラスタ環境で実行している場合はクラスタ上、クライアント コンピュータ上で実行している場合はクライアント コンピュータ上の場所である必要があります。

   > **注：** `path`が指定されていない場合は、現在の作業ディレクトリに新しい CASS レポートが配置されます。

   `reportType`パラメータの値は、**列挙 UAMCASSReportType**（210ページ）に記載された値でなければなりません。1 つ以上のレポート タイプをこのパラメータに指定できます。

Validate Address Global

API エンティティ

GlobalAddressingDetail<T extends ProcessType>

目的
Validate Address Global のジョブの詳細を指定します。

GlobalAddressingEngineConfiguration

目的
Validate Address Global ジョブを作成および実行するために必要なデータベース構成を設定する。

GlobalAddressingFactory

目的
Validate Address Global ジョブのインスタンスを作成するためのシングルトン ファクトリ クラス。

GlobalAddressingGeneralConfiguration

目的
Validate Address Global ジョブを作成および実行するために必要な JVM 構成を設定します。

GlobalAddressingInputConfiguration

目的
Validate Address Global ジョブを作成および実行するために入力の設定を構成する。

入力パラメータ

パラメータ	説明
Validate Address Global Engine Configuration	次のデータベース構成を設定します。 1. データベース タイプ 2. プリロード タイプ 3. リファレンス データ パス 4. すべての国をサポートするかどうか。サポートしない場合は、サポートする国のリスト。

パラメータ	説明
Validate Address Global Input Configuration	次の入力設定を指定します。 **1.** 結果の州/省タイプ **2.** 処理のマッチング範囲 **3.** 入力の強制国 ISO3 **4.** 入力のデフォルト国 ISO3 **5.** 入力の書式区切り記号 **6.** 結果の書式区切り記号 **7.** 結果に入力を含める **8.** 結果の国タイプ **9.** 処理の最適化レベル **10.** 結果の言語設定 **11.** 処理のモード **12.** 結果のスクリプトの設定 **13.** 結果の最大数 **14.** 結果の大文字と小文字の区別
Validate Address Global General Configuration	JVM 設定を指定します。 **1.** キャッシュ サイズ **2.** 最大スレッド数 **3.** 最大住所オブジェクト数 **4.** 拡張する範囲 **5.** 柔軟な範囲拡張 **6.** トランザクションのログ記録の有効化 **7.** 最大メモリ使用量 (単位: MB)
Unlock Code	データベース内のデータをロック解除します。
Reference Data Path	リファレンス データ パスの詳細を指定します。 **注:** UAM ジョブの場合、リファレンス データは、クラスタ内のローカル データ ノードのみに配置する必要があります。
Job Configurations	ジョブ用の Hadoop 設定 MapReduce ジョブの場合、インスタンスのタイプは**MRJobConfig**（39ページ）である必要があります。Spark ジョブの場合、インスタンスのタイプは**SparkJobConfig**（40ページ）である必要があります。

パラメータ	説明
Input File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の入力テキスト ファイルのパス。 **Record Separator** 入力ファイル内で使用されるレコード区切り文字。 **Field Separator** 入力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **Text Qualifier** 区切り記号付きファイル内のテキスト値を囲むのに使用する文字。 **Header Row Fields** 入力ファイルのヘッダー フィールドの配列。 **Skip First Row** 入力ファイル レコードの読み取り時に、先頭行をスキップするかどうかを示すフラグ。 先頭行がヘッダー行である場合は、これを `true` にする必要があります。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC File Path** Hadoop プラットフォーム上の入力 ORC 形式ファイルのパス。 *共通パラメータ:* **Field Mappings** キー値ペアのマップ。既存の列名をキーとし、対応する出力列名を値としてマッピングします。

パラメータ	説明
Output File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の出力テキスト ファイルのパス。 **Field Separator** 出力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC ファイル パス** Hadoop プラットフォーム上の出力 ORC 形式ファイルのパス。 *共通パラメータ:* **Overwrite** 出力ファイルと同じ名前のファイルが既に存在する場合に、上書きするかどうかを示すフラグ。 **Create Output Header** ヘッダー ファイルを Hadoop サーバー上に作成するかどうかを示すフラグ。
Job Name	ジョブの名前。

出力列

住所データ

1. AddressBlock1-9
2. AddressLine1-6
3. AdministrativeDistrict
4. ApartmentLabel
5. ApartmentNumber
6. BlockName
7. BuildingName
8. City
9. City.AddInfo
10. City.SortingCode

11. Contact
12. Country
13. County
14. FirmName
15. Floor
16. HouseNumber
17. LastLine
18. LeadingDirectional
19. Locality
20. POBox
21. PostalCode
22. PostalCode.AddOn
23. PostalCode.Base
24. Room
25. SecondaryStreet
26. StateProvince
27. StreetName
28. StreetSuffix
29. SubBuilding
30. Suburb
31. Territory
32. TrailingDirectional

元の入力データ

1. AddressLine1.Input
2. AddressLine2.Input
3. AddressLine3.Input
4. AddressLine4.Input
5. AddressLine5.Input
6. AddressLine6.Input
7. City.Input
8. StateProvince.Input
9. PostalCode.Input
10. Contact.Input
11. Country.Input
12. FirmName.Input
13. Street.Input
14. Number.Input
15. Building.Input
16. SubBuilding.Input
17. DeliveryService.Input

重要： 入力フィールド `AddressLine2.Input`、`AddressLine3.Input`、`AddressLine4.Input`、`AddressLine5.Input`、および `AddressLine6.Input` は、クラス `GlobalAddressingInputConfiguration` の `resultIncludeInputs` フィールドが `true` に設定された場合にのみ出力に含まれます。そうでない場合、これらの `AddressLineX.input` フィールドで入力に含まれるもののみが、出力に含まれます。

結果コード

1. AddressType
2. Confidence
3. CountOverflow
4. ElementInputStatus
5. ElementRelevance
6. ElementResultStatus
7. MailabilityScore
8. ModeUsed
9. MultimatchCount
10. ProcessStatus
11. Status
12. Status.Code
13. Status.Description

> **注：** フィールドの説明については、Spectrum™ Technology Platformの『*Addressing ガイド*』のトピック「*Validate Address Global*」を参照してください。

Validate Address Global MapReduce ジョブの使用

1. `GlobalAddressingFactory`のインスタンスを、その静的メソッド `getInstance()` を使用して作成します。
2. Validate Address Global ジョブの入力と出力の詳細を指定します。これには `GlobalAddressingDetail`を指定する`ProcessType` のインスタンスを作成します。このインスタンスは、**MRProcessType**（41ページ）タイプを使用する必要があります。これを行うには、次の手順に従います。
   a) `GlobalAddressingGeneralConfiguration`のインスタンスを作成することによって、JVM の初期化を設定します。
   列挙体 **列挙 CacheSize**（207ページ）、**列挙 RangesToExpand**（207ページ）、および **列挙 FlexibleRangeExpansion**（207ページ） を使用します。
   b) `LocalReferenceDataPath`のインスタンスを作成することによって、リファレンス データ パスの詳細を設定します。

c) 必要なデータベース設定を指定します。これには、前述の `GlobalAddressingEngineConfiguration`インスタンスを引数として渡して、`LocalReferenceDataPath` のインスタンスを作成します。

1. 列挙体 **列挙 PreloadingType**（203ページ）を使用してこのインスタンスのプリロード タイプを設定します。
2. **列挙 DatabaseType**（203ページ）使用してデータベース タイプを設定します。
3. **列挙 CountryCodes**（203ページ）を使用してサポートされる国を設定します。
4. すべての国をサポートする場合は、`isAllCountries`属性を true に設定します。そうでない場合は、**列挙 CountryCodes**（203ページ）の値をコンマで区切ったリストで `supportedCountries` 文字列値に指定します。

d) `GlobalAddressingInputConfiguration`のインスタンスを作成することによって、入力を設定します。

このインスタンスの各種フィールドの値を設定するには、列挙体**列挙 CountryCodes**（203ページ）、**列挙 StateProvinceType**（203ページ）、**列挙 CountryType**（203ページ）、**列挙 PreferredScript**（204ページ）、**列挙 PreferredLanguage**（204ページ）、**列挙 Casing**（204ページ）、**列挙 OptimizationLevel**（205ページ）、**列挙 Mode**（205ページ）、および**列挙 MatchingScope**（205ページ） の該当するものを使用します。

e) データにアンロック キーを `String`の `List` 値として設定します。

f) `GlobalAddressingDetail`のインスタンスを作成します。`JobConfig` タイプのインスタンスと、先ほど作成したアンロック コード値の `List`、`GlobalAddressingEngineConfiguration` インスタンス、および `GlobalAddressingInputConfiguration` インスタンスを引数としてコンストラクタに渡します。

`JobConfig`パラメータは、**MRJobConfig**（39ページ） タイプのインスタンスである必要があります。

1. JVM 初期化構成を設定します。`generalConfiguration`インスタンスの `GlobalAddressingDetail` フィールドを上で作成した `GlobalAddressingGeneralConfiguration` インスタンスに設定します。
2. `inputPath`インスタンスの `GlobalAddressingDetail` フィールドを使用して、入力ファイルの詳細を設定します。

   テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

3. `outputPath`インスタンスの `GlobalAddressingDetail` フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

4. `jobName`インスタンスの `GlobalAddressingDetail` フィールドを使用して、ジョブの名前を設定します。

3. 先ほど作成した `GlobalAddressingFactory`のインスタンスを使用してそのメソッド `createJob()` を呼び出し、MapReduce ジョブを作成します。ここで、上の `GlobalAddressingDetail`のインスタンスを引数として渡します。
   `createJob()`メソッドは、`List` のインスタンスの `ControlledJob` を返します。
4. `JobControl`のインスタンスを使用して、作成したジョブを実行します。
5. MapReduce ジョブの正常実行後にレポート カウンタを表示するには、先ほど作成した `GlobalAddressingFactory`のインスタンスを使用して、そのメソッド `getCounters()` を呼び出します。作成したジョブを引数として渡します。

Validate Address Global Spark ジョブの使用

1. `GlobalAddressingFactory`のインスタンスを、その静的メソッド `getInstance()` を使用して作成します。
2. Validate Address Global ジョブの入力と出力の詳細を指定します。これには `GlobalAddressingDetail`を指定する`ProcessType` のインスタンスを作成します。このインスタンスは、**SparkProcessType**（41ページ）タイプを使用する必要があります。これを行うには、次の手順に従います。
   a) `GlobalAddressingGeneralConfiguration`のインスタンスを作成することによって、JVM の初期化を設定します。
   列挙体 **列挙 CacheSize**（207ページ）、**列挙 RangesToExpand**（207ページ）、および **列挙 FlexibleRangeExpansion**（207ページ） を使用します。
   b) `LocalReferenceDataPath`のインスタンスを作成することによって、リファレンス データ パスの詳細を設定します。
   c) 必要なデータベース設定を指定します。これには、前述の `GlobalAddressingEngineConfiguration`インスタンスを引数として渡して、`LocalReferenceDataPath` のインスタンスを作成します。
      1. 列挙体 **列挙 PreloadingType**（203ページ）を使用してこのインスタンスのプリロード タイプを設定します。
      2. **列挙 DatabaseType**（203ページ）使用してデータベース タイプを設定します。
      3. **列挙 CountryCodes**（203ページ）を使用してサポートされる国を設定します。

4. すべての国をサポートする場合は、`isAllCountries`属性を true に設定します。そうでない場合は、**列挙 CountryCodes**（203ページ）の値をコンマで区切ったリストで `supportedCountries` 文字列値に指定します。

d) `GlobalAddressingInputConfiguration`のインスタンスを作成することによって、入力を設定します。

このインスタンスの各種フィールドの値を設定するには、列挙体**列挙 CountryCodes**（203ページ）、**列挙 StateProvinceType**（203ページ）、**列挙 CountryType**（203ページ）、**列挙 PreferredScript**（204ページ）、**列挙 PreferredLanguage**（204ページ）、**列挙 Casing**（204ページ）、**列挙 OptimizationLevel**（205ページ）、**列挙 Mode**（205ページ）、および**列挙 MatchingScope**（205ページ） の該当するものを使用します。

e) データにアンロック キーを `String`の `List` 値として設定します。

f) `GlobalAddressingDetail`のインスタンスを作成します。`JobConfig` タイプのインスタンスと、先ほど作成したアンロック コード値の `List`、`GlobalAddressingEngineConfiguration` インスタンス、および `GlobalAddressingInputConfiguration` インスタンスを引数としてコンストラクタに渡します。

`JobConfig`パラメータは、**SparkJobConfig**（40ページ） タイプのインスタンスである必要があります。

1. JVM 初期化構成を設定します。`generalConfiguration`インスタンスの `GlobalAddressingDetail` フィールドを上で作成した `GlobalAddressingGeneralConfiguration` インスタンスに設定します。
2. `inputPath`インスタンスの `GlobalAddressingDetail` フィールドを使用して、入力ファイルの詳細を設定します。

   テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

3. `outputPath`インスタンスの `GlobalAddressingDetail` フィールドを使用して、出力ファイルの詳細を設定します。

   テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

4. `jobName`インスタンスの `GlobalAddressingDetail` フィールドを使用して、ジョブの名前を設定します。

3. Spark ジョブを作成して実行するには、先ほど作成した `GlobalAddressingFactory`のインスタンスを使用してそのメソッド`runSparkJob()` を呼び出します。ここで、上の `GlobalAddressingDetail`のインスタンスを引数として渡します。

   `runSparkJob()`メソッドはジョブを実行し、ジョブのレポート カウンタの`Map` を返します。
4. カウンタを表示することにより、ジョブに対する統計レポートを表示します。

Validate Address Loqate

API エンティティ

LoqateAddressingDetail<T extends ProcessType>

目的
Validate Address Loqate ジョブの詳細を指定します。

LoqateAddressingEngineConfiguration

目的
Validate Address Loqate ジョブを作成および実行するために必要なデータベース構成を設定する。

LoqateAddressingFactory

目的
Validate Address Loqate ジョブのインスタンスを作成するためのシングルトン ファクトリ クラス。

LoqateAddressingGeneralConfiguration

目的
Validate Address Loqate ジョブを作成および実行するために必要な JVM 構成を設定します。

LoqateAddressingValidateConfiguration

目的
Validate Address Loqate ジョブを作成および実行するために入力の設定を構成する。

入力パラメータ

パラメータ	説明
Validate Address Loqate のエンジン設定	検証を実行するための設定を指定します。 1. 詳細表示 2. ツール情報 3. 住所フォーマットを出力 4. 入力をログに記録 5. 出力をログに記録 6. ログ ファイル名 7. マッチ スコアの絶対しきい値 8. マッチ スコアのしきい値係数 9. 郵便番号の最大結果数 10. 厳密な参照一致
Validate Address Loqate の検証設定	次の入力設定を指定します。 1. 標準住所を含める 2. 一致した住所要素を含める 3. 正規化された入力住所要素 4. 住所データ ブロックを返す 5. 出力の大文字と小文字の区別 6. 個々のフィールドの結果コードを含める 7. 複数の住所を返す 8. 複数の一致が見つかると失敗 9. 複数住所カウント 10. 国フォーマット 11. デフォルト国 12. スクリプト アルファベット 13. ジオコード アドレス フィールドを返す 14. 許容レベル 15. 最小マッチ スコア 16. AMAS 表記を使用してデータをフォーマット 17. 重複処理である 18. 単一フィールドの重複処理 19. 複数フィールドの重複処理 20. 非標準フィールドの重複処理 21. 出力フィールドの重複処理

パラメータ	説明
Validate Address Loqate の全般的な設定	JVM 設定を指定します。 1. 最大アイドル オブジェクト数 2. 最小アイドル オブジェクト数 3. 最大アクティブ オブジェクト数 4. 最長待ち時間 5. 枯渇時のアクション 6. 借用時のテスト 7. 復帰時のテスト 8. アイドル中のテスト 9. 退出の実行間隔 (ミリ秒) 10. 退出実行 1 回あたりのテスト数 11. 最小退出可能アイドル時間 (ミリ秒)
Reference Data Path	リファレンス データ パスの詳細を指定します。 **注:** UAM ジョブの場合、リファレンス データは、クラスタ内のローカル データ ノードのみに配置する必要があります。
Job Configurations	ジョブ用の Hadoop 設定 MapReduce ジョブの場合、インスタンスのタイプは**MRJobConfig**（39ページ）である必要があります。Spark ジョブの場合、インスタンスのタイプは**SparkJobConfig**（40ページ）である必要があります。

パラメータ	説明
Input File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の入力テキスト ファイルのパス。 **Record Separator** 入力ファイル内で使用されるレコード区切り文字。 **Field Separator** 入力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **Text Qualifier** 区切り記号付きファイル内のテキスト値を囲むのに使用する文字。 **Header Row Fields** 入力ファイルのヘッダー フィールドの配列。 **Skip First Row** 入力ファイル レコードの読み取り時に、先頭行をスキップするかどうかを示すフラグ。 先頭行がヘッダー行である場合は、これを `true` にする必要があります。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC File Path** Hadoop プラットフォーム上の入力 ORC 形式ファイルのパス。 *共通パラメータ:* **Field Mappings** キー値ペアのマップ。既存の列名をキーとし、対応する出力列名を値としてマッピングします。

パラメータ	説明
Output File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の出力テキスト ファイルのパス。 **Field Separator** 出力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC ファイル パス** Hadoop プラットフォーム上の出力 ORC 形式ファイルのパス。 *共通パラメータ:* **Overwrite** 出力ファイルと同じ名前のファイルが既に存在する場合に、上書きするかどうかを示すフラグ。 **Create Output Header** ヘッダー ファイルを Hadoop サーバー上に作成するかどうかを示すフラグ。
Job Name	ジョブの名前。

出力列

1. AdditionalInputData
2. AddressLine1-4
3. City
4. Country
5. FirmName
6. PostalCode
7. PostalCode.AddOn
8. PostalCode.Base
9. StateProvince
10. AddressBlock1-9
11. ApartmentLabel
12. ApartmentNumber

13. ApartmentNumber2
14. Building
15. City
16. Country
17. County *
18. FirmName
19. HouseNumber
20. LeadingDirectional
21. POBox
22. PostalCode
23. Principality *
24. StateProvince
25. StreetAlias
26. StreetName
27. StreetSuffix
28. Subcity *
29. Substreet *
30. TrailingDirectional
31. ApartmentLabel.Input
32. ApartmentNumber.Input
33. City.Input
34. Country.Input
35. County.Input *
36. FirmName.Input
37. HouseNumber.Input
38. LeadingDirectional.Input
39. POBox.Input
40. PostalCode.Input
41. Principality.Input *
42. StateProvince.Input
43. StreetAlias.Input
44. StreetName.Input
45. StreetSuffix.Input
46. Subcity.Input *
47. Substreet.Input *
48. TrailingDirectional.Input
49. Geocode.MatchCode
50. Latitude
51. Longitude
52. SearchDistance
53. Confidence

54. CouldNotValidate
55. MatchScore
56. ProcessedBy
57. Status
58. Status.Code
59. Status.Description
60. ApartmentLabel.Result
61. ApartmentNumber.Result
62. City.Result
63. Country.Result
64. County.Result *
65. FirmName.Result
66. HouseNumber.Result
67. LeadingDirectional.Result
68. POBox.Result
69. PostalCode.Result
70. PostalCode.Type
71. Principality.Result *
72. StateProvince.Result
73. StreetAlias.Result
74. StreetName.Result
75. StreetSuffix.Result
76. Subcity.Result *
77. Substreet.Result *
78. TrailingDirectional.Result
79. Barcode
80. DPID
81. FloorNumber
82. FloorType
83. PostalBoxNum

*これはサブフィールドであり、データを含まない場合があります。

表 1：都市/ストリート/郵便番号セントロイド マッチ コード

要素	マッチ コード
住所ポイント	P4
住所ポイント補間済み	I4

要素	マッチ コード
ストリートセントロイド	A4/P3
郵便番号/都市セントロイド	A3/P2/A2

注： フィールドの説明については、Spectrum™ Technology Platformの『*Addressing ガイド*』のトピック「*Validate Address Loqate*」を参照してください。

Validate Address Loqate MapReduce ジョブの使用

1. `LoqateAddressingFactory`のインスタンスを、その静的メソッド `getInstance()` を使用して作成します。
2. `LoqateAddressingDetail`を指定する`ProcessType` のインスタンスを作成して、Validate Address Loqate ジョブの入力と出力の詳細を指定します。このインスタンスは、**MRProcessType**（41ページ）タイプを使用する必要があります。これを行うには、次の手順に従います。
   a) `LoqateAddressingGeneralConfiguration`のインスタンスを作成することによって、JVM の初期化を設定します。
   列挙体 **列挙 ExhaustedAction**（206ページ）を使用します。
   b) `LoqateAddressingEngineConfiguration`のインスタンスを作成して必要なデータベース設定を行い、各種フィールドを設定します。
   c) `LoqateAddressingValidateConfiguration`のインスタンスを作成して、住所検証の設定を行います。
   このインスタンスの各種フィールドの値を設定するには、列挙体 **列挙 AcceptanceLevel**（206ページ）、**列挙 CountryCodes**（203ページ）、**列挙 OutputCasing**（206ページ）、**列挙 CountryFormat**（206ページ）、および **列挙 ScriptAlphabet**（206ページ） を使用します。
   d) `LocalReferenceDataPath`のインスタンスを作成することによって、リファレンス データ パスの詳細を設定します。
   e) `LoqateAddressingDetail`のインスタンスを作成します。`JobConfig` タイプのインスタンス、`LocalReferenceDataPath` インスタンス、および先ほど作成した `LoqateAddressingValidateConfiguration` インスタンスを引数としてコンストラクタに渡します。
   `JobConfig`パラメータは、**MRJobConfig**（39ページ） タイプのインスタンスである必要があります。

1. `inputPath`インスタンスの `LoqateAddressingDetail` フィールドを使用して、入力ファイルの詳細を設定します。

   テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

2. `outputPath`インスタンスの `LoqateAddressingDetail` フィールドを使用して、出力ファイルの詳細を設定します。

   テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

3. `jobName`インスタンスの `LoqateAddressingDetail` フィールドを使用して、ジョブの名前を設定します。

3. 先ほど作成した `LoqateAddressingFactory`のインスタンスを使用してそのメソッド `createJob()` を呼び出し、MapReduce ジョブを作成します。ここで、上の `LoqateAddressingDetail`のインスタンスを引数として渡します。`createJob()`メソッドは、`List` のインスタンスの `ControlledJob` を返します。
4. `JobControl`のインスタンスを使用して、作成したジョブを実行します。
5. MapReduce ジョブの正常実行後にレポート カウンタを表示するには、先ほど作成した `LoqateAddressingFactory`のインスタンスを使用して、そのメソッド `getCounters()` を呼び出します。作成したジョブを引数として渡します。

Validate Address Loqate Spark ジョブの使用

1. `LoqateAddressingFactory`のインスタンスを、その静的メソッド `getInstance()` を使用して作成します。
2. `LoqateAddressingDetail`を指定する`ProcessType` のインスタンスを作成して、Validate Address Loqate ジョブの入力と出力の詳細を指定します。このインスタンスは、**SparkProcessType**（41ページ）タイプを使用する必要があります。これを行うには、次の手順に従います。

   a) `LoqateAddressingGeneralConfiguration`のインスタンスを作成することによって、JVM の初期化を設定します。

   列挙体 **列挙 ExhaustedAction**（206ページ）を使用します。

   b) `LoqateAddressingEngineConfiguration`のインスタンスを作成して必要なデータベース設定を行い、各種フィールドを設定します。

c) `LoqateAddressingValidateConfiguration`のインスタンスを作成して、住所検証の設定を行います。

このインスタンスの各種フィールドの値を設定するには、列挙体 **列挙 AcceptanceLevel**（206ページ）、**列挙 CountryCodes**（203ページ）、**列挙 OutputCasing**（206ページ）、**列挙 CountryFormat**（206ページ）、および **列挙 ScriptAlphabet**（206ページ） を使用します。

d) `LocalReferenceDataPath`のインスタンスを作成することによって、リファレンス データ パスの詳細を設定します。

e) `LoqateAddressingDetail`のインスタンスを作成します。`JobConfig` タイプのインスタンス、`LocalReferenceDataPath` インスタンス、および先ほど作成した `LoqateAddressingValidateConfiguration` インスタンスを引数としてコンストラクタに渡します。

`JobConfig`パラメータは、**SparkJobConfig**（40ページ） タイプのインスタンスである必要があります。

1. `inputPath`インスタンスの `LoqateAddressingDetail` フィールドを使用して、入力ファイルの詳細を設定します。

   テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

2. `outputPath`インスタンスの `LoqateAddressingDetail` フィールドを使用して、出力ファイルの詳細を設定します。

   テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定して`FilePath`のインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して `OrcFilePath`のインスタンスを作成します。

3. `jobName`インスタンスの `LoqateAddressingDetail` フィールドを使用して、ジョブの名前を設定します。

3. Spark ジョブを作成して実行するには、先ほど作成した `LoqateAddressingFactory`のインスタンスを使用してそのメソッド`runSparkJob()` を呼び出します。ここで、上の `LoqateAddressingDetail`のインスタンスを引数として渡します。

   `runSparkJob()`メソッドはジョブを実行し、ジョブのレポート カウンタの`Map` を返します。

4. カウンタを表示することにより、ジョブに対する統計レポートを表示します。

Universal Name モジュールのジョブ

コモンモジュール API

UniversalNameDetail<T extends ProcessType>

目的
Universal Name モジュールのジョブの詳細を指定します。

UniversalNameFactory

目的
Universal Name モジュールのジョブのインスタンスを作成するためのシングルトン ファクトリ クラス。

Open Name Parser

API エンティティ

OpenNameParserDetail

目的
Open Name Parser ジョブの詳細を指定します。

OpenNameParserConfiguration

目的
`name` データ フィールドにある個人名、企業名、またはその他の名称を構成要素に分解します。

入力パラメータ

パラメータ	説明
Open Name Parser Configuration	name データ フィールドにある個人名、企業名、またはその他の名称を構成要素に分解します。
Reference Data Path	リファレンス データ パスの詳細を指定します。
Job Configurations	ジョブ用の Hadoop 設定 MapReduce ジョブの場合、インスタンスのタイプは**MRJobConfig**（39ページ）である必要があります。Spark ジョブの場合、インスタンスのタイプは **SparkJobConfig**（40ページ）である必要があります。

パラメータ	説明
Input File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の入力テキスト ファイルのパス。 **Record Separator** 入力ファイル内で使用されるレコード区切り文字。 **Field Separator** 入力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **Text Qualifier** 区切り記号付きファイル内のテキスト値を囲むのに使用する文字。 **Header Row Fields** 入力ファイルのヘッダー フィールドの配列。 **Skip First Row** 入力ファイル レコードの読み取り時に、先頭行をスキップするかどうかを示すフラグ。 先頭行がヘッダー行である場合は、これを `true` にする必要があります。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC File Path** Hadoop プラットフォーム上の入力 ORC 形式ファイルのパス。 *共通パラメータ:* **Field Mappings** キー値ペアのマップ。既存の列名をキーとし、対応する出力列名を値としてマッピングします。

パラメータ	説明
Output File	*テキスト ファイルの場合:* **File Path** Hadoop プラットフォーム上の出力テキスト ファイルのパス。 **Field Separator** 出力ファイルで、レコード内の連続する 2 つのフィールドの間に使用される区切り文字。 **重要：** `FilePath`の適切なコンストラクタを呼び出します。 *ORC 形式ファイル:* **ORC ファイル パス** Hadoop プラットフォーム上の出力 ORC 形式ファイルのパス。 *共通パラメータ:* **Overwrite** 出力ファイルと同じ名前のファイルが既に存在する場合に、上書きするかどうかを示すフラグ。 **Create Output Header** ヘッダー ファイルを Hadoop サーバー上に作成するかどうかを示すフラグ。
Job Name	ジョブの名前。

出力列

入力列に加えて、Open Name Parser ジョブの出力生成時に以下の列が追加されます。

	書式	説明
AccountDescription	String	名前の一部であるアカウント説明。例えば、"Mary Jones Account # 12345" で、アカウント説明は "Account#12345"。
会社名関係のフィールド		

	書式	説明
FirmConjunction	String	"d/b/a" (doing business as)、"o/a" (operating as)、"t/a" (trading as) などの略語を含む企業の名前を示します。
FirmName	String	会社名。例えば、"Pitney Bowes"。
FirmSuffix	String	会社名の接尾語。例えば、"Co."、"Inc."
IsFirm	String	名前が、個人名ではなく、企業名であることを示します。値は True または False です。
個人名に関係するフィールド		
Conjunction	String	名前に、"and"、"or"、"&" などの接続詞が含まれることを示します。
CultureCode	String	入力データに含まれるカルチャー コード。
CultureCodeUsedToParse	String	データのパースに使用されたカルチャー固有のグラマーを特定します。 **Null (empty)** グローバル カルチャー (デフォルト)。 **de** ドイツ語。 **es** スペイン語。 **ja** 日本語。
FirstName	String	個人のファースト ネーム。
GeneralSuffix	String	個人名の一般/職業接尾語。例えば、 MD PhD 。
IsParsed	String	出力レコードがパースされたかどうかを示します。値は True または False です。

	書式	説明
IsPersonal	String	名前が企業名ではなく、個人名であるかどうかを示します。値は True または False です。
IsReverseOrder	String	入力名が逆順序であるかどうかを示します。値は True または False です。
LastName	String	個人名のラスト ネーム。父方の姓が含まれます。
LeadingData	String	名前の前に付けられる、名前以外の情報。
MaturitySuffix	String	個人の世代/家族接尾語。例えば、Jr.または Sr.。
MiddleName	String	個人のミドル ネーム。
Name.	String	入力に指定された個人名または企業名。
NameScore	String	各名前の既知および不明トークンの平均スコアを示します。NameScore の値は、パーシング グラマーでの定義に従って、0 ～ 100 の間になります。マッチが返されない場合は、0 が返されます。
SecondaryLastName	String	スペイン語のパーシング グラマーでは、その人の母の姓。
TitleOfRespect	String	"Mr."、"Mrs."、"Dr." など、名前の前に付けられる情報。
TrailingData	String	名前の後に付けられる、名前以外の情報。
結合名関係のフィールド		

	書式	説明
Conjunction2	String	結合されている 2 番目の名前に、"and"、"or"、"&" などの接続詞が含まれることを示します。
Conjunction3	String	結合されている 3 番目の名前に、"and"、"or"、"&" などの接続詞が含まれることを示します。
FirmName2	String	結合されている 2 番目の企業名。例えば、Baltimore Gas & Electric dba Constellation Energy。
FirmSuffix2	String	結合されている 2 番目の企業の接尾語。
FirstName2	String	結合されている FirstName の 2 番目の名
FirstName3	String	結合されている FirstName の 3 番目の名。
GeneralSuffix2	String	結合されている 2 番目の名前の一般/職業接尾語。例えば、 MD PhD 。
GeneralSuffix3	String	結合されている 3 番目の名前の一般/職業接尾語。例えば、 MD PhD 。
IsConjoined	String	入力名が結合名であることを示します。結合名は、例えば、"John and Jane Smith"。値は True または False です。
LastName2	String	結合されている LastName の 2 番目の姓。
LastName3	String	結合されている LastName の 3 番目の姓。
MaturitySuffix2	String	結合されている 2 番目の名前の世代/家族接尾語。例えば、Jr.または Sr.。

	書式	説明
MaturitySuffix3	String	結合されている 3 番目の名前の世代/家族接尾語。例えば、Jr.または Sr.。
MiddleName2	String	結合されている MiddleName の 2 番目の名。
MiddleName3	String	結合されている MiddleName の 3 番目の名。
TitleOfRespect2	String	"Mr."、"Mrs."、"Dr." など、結合されている 2 番目の名前の前に付けられる情報。
TitleOfRespect3	String	"Mr."、"Mrs."、"Dr." など、結合されている 3 番目の名前の前に付けられる情報。

Open Name Parser MapReduce ジョブの使用

1. `UniversalNameFactory`のインスタンスを、その静的メソッド `getInstance()` を使用して作成します。
2. Open Name Parser ジョブの入力と出力の詳細を指定します。以下の手順に従って、`OpenNameParserDetail`を指定する`ProcessType` のインスタンスを作成することによって、これを行います。このインスタンスは、**MRProcessType**（41ページ）タイプを使用する必要があります。
   a) `OpenNameParserConfiguration`のインスタンスを作成することによって、Open Name Parser ルールを設定します。
   b) `ReferenceDataPath`のインスタンスを作成することによって、リファレンス データ パスと場所のタイプの詳細を設定します。列挙 **ReferenceDataPathLocation**（200ページ）を参照してください。
   c) `OpenNameParserDetail`のインスタンスを作成します。`JobConfig` タイプのインスタンスと、上で作成した `OpenNameParserConfiguration` と `ReferenceDataPath` のインスタンスを、コンストラクタの引数として渡します。
   `JobConfig`パラメータは、**MRJobConfig**（39ページ） タイプのインスタンスである必要があります。
   d) `inputPath`インスタンスの `OpenNameParserDetail` フィールドを使用して、入力ファイルの詳細を設定します。

テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

e) outputPathインスタンスの OpenNameParserDetail フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

f) jobNameインスタンスの OpenNameParserDetail フィールドを使用して、ジョブの名前を設定します。

3. 先ほど作成したUniversalNameFactoryのインスタンスを使用してそのメソッドcreateJob()を呼び出し、MapReduce ジョブを作成します。ここで、上の OpenNameParserDetailのインスタンスを引数として渡します。
createJob()メソッドは、List のインスタンスの ControlledJob を返します。
4. JobControlのインスタンスを使用して、作成したジョブを実行します。
5. MapReduce ジョブの正常実行後にレポート カウンタを表示するには、先ほど作成したUniversalNameFactoryのインスタンスを使用して、そのメソッド getCounters()を呼び出します。作成したジョブを引数として渡します。

Open Name Parser Spark ジョブの使用

1. UniversalNameFactoryのインスタンスを、その静的メソッド getInstance()を使用して作成します。
2. Open Name Parser ジョブの入力と出力の詳細を指定します。以下の手順に従って、ProcessType を指定する OpenNameParserDetail のインスタンスを作成することによって、これを行います。このインスタンスは、**SparkProcessType**（41ページ） タイプを使用する必要があります。
   a) OpenNameParserConfigurationのインスタンスを作成することによって、Open Name Parser ルールを設定します。
   b) ReferenceDataPathのインスタンスを作成することによって、リファレンス データ パスと場所のタイプの詳細を設定します。**列挙 ReferenceDataPathLocation**（200ページ）を参照してください。
   c) OpenNameParserDetail のインスタンスを作成します。JobConfig タイプのインスタンスと、上で作成した OpenNameParserConfiguration と ReferenceDataPath のインスタンスを、コンストラクタの引数として渡します。

JobConfig パラメータは、**SparkJobConfig**（40ページ） タイプのインスタンスである必要があります。

d) inputPathインスタンスの OpenNameParserDetail フィールドを使用して、入力ファイルの詳細を設定します。

テキスト入力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な入力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 入力ファイルの場合、ORC 入力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

e) outputPathインスタンスの OpenNameParserDetail フィールドを使用して、出力ファイルの詳細を設定します。

テキスト出力ファイルの場合は、適切なコンストラクタを呼び出して、関連する詳細な出力ファイル情報を指定してFilePathのインスタンスを作成します。ORC 出力ファイルの場合、ORC 出力ファイルのパスを引数に指定して OrcFilePathのインスタンスを作成します。

f) jobNameインスタンスの OpenNameParserDetail フィールドを使用して、ジョブの名前を設定します。

3. Spark ジョブを作成して実行するには、先ほど作成した UniversalNameFactory のインスタンスを使用してそのメソッド runSparkJob() を呼び出します。ここで、上の OpenNameParserDetail のインスタンスを引数として渡します。

   runSparkJob() メソッドはジョブを実行し、ジョブのレポート カウンタの Map を返します。

4. カウンタを表示することにより、ジョブに対する統計レポートを表示します。

5 - Hive ユーザ定義関数

このセクションの構成

はじめに

Apache Hive は、ユーザ定義関数 (UDF) を提供します。UDF を定義して、必要なアクションを実行し、所望の目的を達成することができます。

Big Data Quality SDKでは、以下のデータ品質ジョブを実行するための一連の Hive ユーザ定義関数とユーザ定義集約関数が提供されています。

ユーザ定義関数 (UDF)
ユーザ定義関数は、一度に 1 つのレコードを処理します。

UDF に基づくジョブには以下のものがあります。

- Match Key Generator
- Table Lookup
- Advanced Transformer
- Open Name Parser

ユーザ定義集約関数 (UDAF)
ユーザ定義集約関数は、結合フィールドに基づいてレコードをコレクションに集約してから、一度に 1 つのレコード コレクションを処理します。

UDAF に基づくジョブには以下のものがあります。

- Interflow Match
- Intraflow Match
- Transactional Match
- Best of Breed
- Duplicate Synchronization
- Filter
- Validate Address
- Validate Address Global
- Validate Address Loqate

Big Data Quality SDK Hive 関数のコンポーネント

Big Data Quality SDK Hive UDF の実行に必要な主要コンポーネントは、以下のとおりです。

JAR ファイル 必要なデータ品質 Hive UDF が属するモジュールのBig Data Quality SDKHive JAR ファイル。いずれかの UDF を使用する前に、これが登録されている必要があります。

ジョブ UDF / UDAF 各データ品質ジョブは、ユーザ定義関数 (UDF) またはユーザ定義集約関数 (UDAF) として提供されます。

エイリアス Hive UDF に割り当てられたエイリアス。この手順は省略可能です。

設定 実行するジョブに基づく、JSON 形式で指定されたルールとその他の環境設定詳細情報。

ヘッダー 入力テーブルのヘッダ フィールド (カンマ区切り形式)。

入力テーブル 実行する Hive UDF ごとに入力レコードを提供するテーブル。

候補テーブル Interflow Match UDAF の場合、実行する Hive UDF に候補レコードを提供するテーブル。

サスペクト テーブル Interflow Match UDAF の場合、実行する Hive UDF にサスペクト レコードを提供するテーブル。

Hive.Map.Aggr Mapper および Reducer 間でのデータの集約をオンまたはオフにするには、この Hive 環境変数を`false`に設定します。デフォルトでは、`Hive.Map.Aggr = true`となっており、データは集約されます。

SDK 内のすべての Hive ジョブで、この値を `false` に設定します。

> **注 :** この設定はすべての UDAF で必要です。

全般的な設定 ジョブを実行するために必要なメモリ設定。

> **注 :** この設定は Universal Addressing モジュールの Hive UDAF でのみ必要です。

入力設定 入力データの設定。

> **注 :** この設定は Universal Addressing モジュールの Hive UDAF でのみ必要です。

エンジン設定 データベース設定、*COBOL ランタイム パス*、*プリロード タイプ*など、さまざまな設定を行います。

> **注 :** この設定は Universal Addressing モジュールの Hive UDAF でのみ必要です。

LD_LIBRARY_PATH この環境変数は、Hive ジョブの実行時に必要なさまざまな COBOL ライブラリへのパスに設定します。

> **注：** この設定は Validate Address の Hive UDAF でのみ必要です。

プロセス タイプ SDK の特定の Hive ジョブで使用される適切な検証レベルを指定します。現時点では、住所検証のみがサポートされています。

この値は `VALIDATE` に設定します。

> **注：** この設定は、Validate Address および Validate Address Loqate の Hive UDAF でのみ必要です。

出力 Hive UDF の出力。コンソールに表示されるか、出力ファイルに書き出されます。

クエリ 必要な Hive UDF を実行するクエリ。

各ジョブに対し、適切なクエリ構文を用いて以下の操作を実行できます。

- ジョブの出力をコンソールに表示する。
- 指定された出力ファイルに出力を書き出す。

Hive UDF の使用

Hive UDF ベースの各ジョブを実行するには、Hive クライアント上で 1 つのセッションで以下のステップを個別に実行するか、必要なすべてのステップをまとめた HQL ファイルを作成してそれを一度に実行することができます。

1. Hive クライアントで、必要な Hive データベースにログインします。
2. 必要なデータ品質 Hive UDF が属する特定の Big Data Quality SDKモジュールの JAR ファイルを登録します。
3. Validate Address の UDAF の場合、COBOL ライブラリのパスを設定するには、環境変数 `LD_LIBRARY_PATH`を次のように設定します。

```
set mapreduce.admin.user.env =
LD_LIBRARY_PATH=/home/hduser/~/runtime/lib:
/home/hduser/~/runtime/bin:/home/hduser/~/server/modules/universaladdress/lib,
 ACU_RUNCBL_JNI_ONLOAD_DISABLE=1, G1RTS=/home/hduser/~/ ;
```

4. Validate Address Global の UDAF の場合は、*libAddressDoctor5.so* ファイルも追加します。

5. Validate Address Loqate の UDAF の場合は、以下の必須ファイルを分散キャッシュに追加します。
   - loqate-core.car
   - LoqateVerificationLevel.csv
   - Loqate.csv
   - countryTables.csv
   - countryNameTables.csv
6. 実行するデータ品質ジョブの Hive UDF のエイリアスを作成します。
   例:

```
CREATE TEMPORARY FUNCTION matchkeygenerator as
'com.pb.bdq.amm.process.hive.matchkeygenerator.MatchKeyGeneratorUDF';
```

7. マッチ ルール、ソート フィールド、Express マッチ列といったジョブの詳細情報を環境設定として指定し、それぞれの変数または設定プロパティに代入します。

   注 : ルールは JSON 形式である必要があります。

   例:

```
set rule='{"matchKeys":[{"expressMatchKey":false,
"matchKeyField":"MatchKey1",
"rules":[{"algorithm":"Soundex"," field":"businessname",
"startPosition":1, "length":0,"active":true, "sortInput":null,
"removeNoiseCharacters":false}]},
{"expressMatchKey":false, "matchKeyField":"MatchKey2",
"rules":[{"algorithm":"Koeln", "field":"businessname",
"startPosition":1, "length":0, "active":true, "sortInput":null,
 "removeNoiseCharacters":false}]}]}';
```

   注 : それぞれのジョブ環境設定の設定プロパティを、必ず使用してください。例えば、それぞれのサンプル HQL ファイルに記載されている `pb.bdq.match.rule`、`pb.bdq.match.express.column`、`pb.bdq.consolidation.sort.field` などです。

8. 入力テーブルのヘッダ フィールドをカンマ区切り形式で指定し、変数または設定プロパティに割り当てます。

```
set pb.bdq.match.header='businessname,recordid';
```

   注 : 記載されている設定プロパティを必ず使用してください。例えば、それぞれのサンプル HQL ファイルに記載されている `pb.bdq.match.header`、`pb.bdq.consolidation.header` などです。

9. 次の例に示すように、Hive.Map.Aggr環境変数の設定を false にして、Reducer および Mapper 間でのデータの集約をオフにします。

```
set hive.map.aggr = false;
```

注： この設定はすべての UDAF で必要です。

10. ジョブを実行するための全般的な設定を次の例に示すように設定します。

```
set pb.bdq.uam.universaladdress.general.configuration =
{"dFileType":"SPLIT", "dMemoryModel":"MEDIUM",
"lacsLinkMemoryModel":"MEDIUM", "suiteLinkMemoryModel":"MEDIUM"};
```

注： この設定は Universal Addressing モジュールの Hive UDAF でのみ必要です。

11. ジョブを実行するための入力設定を次の例に示すように設定します。

```
set pb.bdq.uam.universaladdress.input.configuration =
{"outputStandardAddress":true, "outputPostalData":false,
"outputParsedInput":false, "outputAddressBlocks":true,
"performUSProcessing":true, "performCanadianProcessing":false,
"performInternationalProcessing":false, "outputFormattedOnFail":false,
 "outputCasing":"MIXED", "outputPostalCodeSeparator":true,
"outputMultinationalCharacters":false, "performDPV":false,
"performRDI":false, "performESM":false, "performASM":false,
"performEWS":false, "performLACSLink":false, "performLOT":false,
"failOnCMRAMatch":false, "extractFirm":false, "extractUrb":false,
"outputReport3553":false, "outputReportSERP":false,
"outputReportSummary":true, "outputCASSDetail":false,
"outputFieldLevelReturnCodes":false, "keepMultimatch":false,
"maximumResults":10,
"standardAddressFormat":"STANDARD_ADDRESS_FORMAT_COMBINED_UNIT",
"standardAddressPMBLine":"STANDARD_ADDRESS_PMB_LINE_NONE",
"cityNameFormat":"CITY_FORMAT_STANDARD", "vanityCityFormatLong":true,
 "outputCountryFormat":"ENGLISH", "homeCountry":"United States",
"streetMatchingStrictness":"MATCHING_STRICTNESS_MEDIUM",
"firmMatchingStrictness":"MATCHING_STRICTNESS_MEDIUM",
"directionalMatchingStrictness":"MATCHING_STRICTNESS_MEDIUM",
"dualAddressLogic":"DUAL_NORMAL", "dpvSuccessfulStatusCondition":"A",
 "reportListFileName":"", "reportlistProcessorName":"",
"reportlistNumber":1, "reportMailerAddress":"", "reportMailerName":"",
 "reportMailerCityLine":"", "canReportMailerCPCNumber":"",
"canReportMailerAddress":"", "canReportMailerName":"",
"canReportMailerCityLine":"", "internationalCityStreetSearching":100,
 "addressLineSearchOnFail":true, "outputStreetAlias":true,
"outputVeriMoveBlock":false, "dpvDetermineNoStat":false,
"dpvDetermineVacancy":false, "outputAbbreviatedAlias":false,
"outputPreferredAlias":false,
"outputPreferredCity":"CITY_OVERRIDE_NAME_ZIP4",
```

```
"performSuiteLink":false, "suppressZplusPhantomCarrierR777":false,
"canStandardAddressFormat":"D", "canEnglishApartmentLabel":"APT",
"canFrenchApartmentLabel":"APP", "canFrenchFormat":"C",
"canOutputCityFormat":"D", "canOutputCityAlias":true,
"canDualAddressLogic":"D", "canPreferHouseNum":false,
"canSSLVRFLG":false, "canRuralRouteFormat":"A", "canNonCivicFormat":"A",
 "canDeliveryOfficeFormat":"I", "canEnableSERP":false,
"canSwitchManagedPostalCodeConfidence":false, "stats":null,
"counts":null, "z3seg":null, "serpStats":null, "dpvSeedList":null,
"lacsSeedList":null, "zipInputSet":null, "reportName":null,
"currentUser":null, "jobName":null, "jobId":null, "jobRequest":false,
 "properties":{"DPVDetermineVacancy":"N", "DualAddressLogic":"N",
"ExtractUrb":"N", "CanFrenchFormat":"C", "AddressLineSearchOnFail":"Y",
 "OutputFieldLevelReturnCodes":"N", "OutputFormattedOnFail":"N",
"OutputStreetNameAlias":"Y", "OutputReportSERP":"N",
"OutputAddressBlocks":"Y", "ExtractFirm":"N",
"CanEnglishApartmentLabel":"APT", "OutputPreferredCity":"Z",
"FirmMatchingStrictness":"M", "CanFrenchApartmentLabel":"APP",
"KeepMultimatch":"N", "StandardAddressPMBLine":"N",
"PerformSuiteLink":"N", "CanStandardAddressFormat":"D",
"DPVSuccessfulStatusCondition":"A", "PerformLACSLink":"N",
"PerformUSProcessing":"Y", "PerformEWS":"N",
"StandardAddressFormat":"C", "SuppressZplusPhantomCarrierR777":"N",
"HomeCountry":"United States", "ReportMailerAddress":"",
"OutputReport3553":"N", "OutputVeriMoveDataBlock":"N",
"CanDeliveryOfficeFormat":"I", "OutputAbbreviatedAlias":"N",
"PerformCanadianProcessing":"N", "PerformDPV":"N",
"PerformInternationalProcessing":"N", "CanSSLVRFlg":"N",
"StreetMatchingStrictness":"M",
"InternationalCityStreetSearching":"100",
"canSwitchManagedPostalCodeConfidence":"N", "CanDualAddressLogic":"D",
 "PerformASM":"N", "OutputCasing":"M", "ReportListFileName":"",
"CanReportMailerAddress":"", "ReportMailerCityLine":"",
"CanReportMailerCPCNumber":"", "ReportListProcessorName":"",
"CanOutputCityAlias":"Y", "DirectionalMatchingStrictness":"M",
"CanRuralRouteFormat":"A", "CanOutputCityFormat":"D",
"ReportListNumber":"1", "CanReportMailerCityLine":"",
"OutputMultinationalCharacters":"N", "EnableSERP":"N",
"CanNonCivicFormat":"A", "OutputShortCityName":"S",
"OutputPostalCodeSeparator":"Y", "FailOnCMRAMatch":"N",
"PerformLOT":"N", "OutputCountryFormat":"E", "CanPreferHouseNum":"N",
 "CanReportMailerName":"", "PerformRDI":"N", "ReportMailerName":"",
"PerformESM":"N", "OutputReportSummary":"Y",
"OutputVanityCityFormatLong":"Y", "OutputPreferredAlias":"N",
"DPVDetermineNoStat":"N", "MaximumResults":"10"}}};
```


注： この設定は Universal Addressing モジュールの Hive UDAF でのみ必要です。

12. ジョブを実行するためのエンジン設定を次の例のように設定します。

```
set pb.bdq.uam.universaladdress.engine.configurations = {
"referenceData":{
"dataDir":"/home/hduser/resources/uam/universaladdress/UAM_universaladdress4.0_Feb15/",
 "referenceDataPathLocation":"LocaltoDataNodes"},
"cobolRuntimePath":"/home/hduser/tapan/addressquality/",
"modulesDir":"/home/hduser/tapan/addressquality/modules",
"dpvDbPath":null, "suiteLinkDBPath":null, "ewsDBPath":null,
"rdiDBPath":null, "lacsDBPath":null};
```

注： この設定は Universal Addressing モジュールの Hive UDAF でのみ必要です。

13. 適切な検証レベルを示すようにプロセス タイプを設定します。現時点では住所検証のみがサポートされています。

例えば、*Validate Address* ジョブでは、*プロセス タイプ*を次のように設定します。

```
set pb.bdq.uam.universaladdress.process.type=VALIDATE;
```

注： この設定は、Validate Address および Validate Address Loqate の Hive UDAF でのみ必要です。

14. ジョブを実行してジョブ出力をコンソールに表示するには、以下の例に示すようにクエリを記述します。

```
SELECT businessname, recordid, bar.ret["MatchKey1"] AS MatchKey1,
bar.ret["MatchKey2"] AS MatchKey2 FROM (
SELECT *, matchkeygenerator (${hiveconf:rule}, ${hiveconf:header},
businessname, recordid) AS ret FROM cust ) bar;
```

ジョブを実行してジョブ出力を指定されたファイルに書き出すには、以下の例に示すようにクエリを記述します。

```
INSERT OVERWRITE LOCAL DIRECTORY '/home/hadoop/MatchKey/' row format
delimited FIELDS TERMINATED BY ',' MAP FIELDS TERMINATED BY ':'
COLLECTION ITEMS TERMINATED BY '|' LINES TERMINATED BY '\n' STORED AS
 TEXTFILE
SELECT businessname, recordid, bar.ret["MatchKey1"] AS MatchKey1,
bar.ret["MatchKey2"] AS MatchKey2 FROM (
SELECT *, matchkeygenerator (${hiveconf:rule}, ${hiveconf:header},
businessname, recordid) AS ret FROM cust ) bar;
```

注： 先ほど UDF に対して定義したエイリアスを必ず使用してください。

重要： すべての UDAF ジョブで、それぞれの設定プロパティを変数として使用するとともに、該当するサンプル HQL ファイルに示されている入力パラメータを定義します。

例えば、pb.bdq.match.rule、pb.bdq.match.express.column、pb.bdq.consolidation.sort.field などです。

Advanced Matching モジュールの関数

Match Key Generator

サンプル Hive スクリプト

```
-- Register Advance Matching Module[AMM] Hive UDF jar
ADD JAR <Directory path>/amm.hive.${project.version}.jar;

-- Provide alias to UDF class (optional). String in quotes represent
class names needed for this job to run.
CREATE TEMPORARY FUNCTION matchkeygenerator as
'com.pb.bdq.amm.process.hive.matchkeygenerator.MatchKeyGeneratorUDF';

-- Match Key Generator is implemented as a UDF (User Defined function).
 It processes one row at a time and generates a map of match keys for
each row.

-- Set rule and header
set rule='{"matchKeys":[{"expressMatchKey":false,
"matchKeyField":"MatchKey1",
"rules":[{"algorithm":"Soundex"," field":"businessname",
"startPosition":1, "length":0,"active":true, "sortInput":null,
"removeNoiseCharacters":false}]},
{"expressMatchKey":false, "matchKeyField":"MatchKey2",
"rules":[{"algorithm":"Koeln", "field":"businessname", "startPosition":1,
 "length":0, "active":true, "sortInput":null,
"removeNoiseCharacters":false}]}]}';

set header='businessname,recordid';

-- Execute query on the desired table to display the job output on
console. This query returns a map of key value for each row containing
 matchkeys as per rule passed.
SELECT businessname, recordid, bar.ret["MatchKey1"] AS MatchKey1,
bar.ret["MatchKey2"] AS MatchKey2 FROM (
SELECT *, matchkeygenerator (${hiveconf:rule}, ${hiveconf:header},
businessname, recordid) AS ret FROM cust ) bar;
```

```
-- Query to dump output to a directory in file system
INSERT OVERWRITE LOCAL DIRECTORY '/home/hadoop/MatchKey/' row format
delimited FIELDS TERMINATED BY ',' MAP FIELDS TERMINATED BY ':'
COLLECTION ITEMS TERMINATED BY '|' LINES TERMINATED BY '\n' STORED AS
TEXTFILE
SELECT businessname, recordid, bar.ret["MatchKey1"] AS MatchKey1,
bar.ret["MatchKey2"] AS MatchKey2 FROM (
SELECT *, matchkeygenerator (${hiveconf:rule}, ${hiveconf:header},
businessname, recordid) AS ret FROM cust ) bar;

--Sample data in input table customer
--+-----------------------------------------+----------------+--+
--|            cust.businessname            | cust.recordid  |
--+-----------------------------------------+----------------+--+
--| Internal Revenue Service                | 0              |
--| Juan F Vera-Monroig                     | 1              |
--| Leonardo Pagan-Reyes                    | 2              |
--| Academia San Joaquin Colegios/Academias | 3              |
--| Nereida Portalatin-Padua                | 4              |
--+-----------------------------------------+----------------+--+


--Sample output for input query
+-------------------------------------+----------+-----------+-----------------+--+
|               businessname          | recordid  | matchkey1  |
    matchkey2     |
+-------------------------------------+----------+-----------+-----------------+--+
| Internal Revenue Service            | 0         | I536       |
0627657368738     |
| Juan F Vera-Monroig                 | 1         | J511       |
063376674         |
| Leonardo Pagan-Reyes                | 2         | L563       |
567214678         |
| Academia San Joaquin Colegios/Academias  | 3         | A235       |
0426864645484268  |
| Nereida Portalatin-Padua            | 4         | N631       |
67217252612       |
+-------------------------------------+----------+-----------+-----------------+--+
```

Interflow Match

サンプル Hive スクリプト

```
-- Register Advance Matching Module[AMM] Hive UDF jar
ADD JAR <Directory path>/amm.hive.${project.version}.jar;

-- Provide alias to UDF class (optional). String in quotes represent
```

```
class names needed for this job to run.
CREATE TEMPORARY FUNCTION rowid as
'com.pb.bdq.hive.common.RowIDGeneratorUDF';

-- This rowid is needed by Interflow Match to maintain the order of rows
 while creating groups. This is a UDF (User Defined Function) and
associates an incremental unique integer number to each row of the data.

CREATE TEMPORARY FUNCTION InterMatch as
'com.pb.bdq.amm.process.hive.interflow.InterMatchUDAF';

-- Inter Flow is implemented as a UDAF (User Defined Aggregation
function). It processes one group of rows at a time based on join field
 and generates the result for that group of rows.

-- Disable map side aggregation
set hive.map.aggr = false;

-- Set the rule using configuration property 'pb.bdq.match.rule'

set pb.bdq.match.rule={"type":"Parent",
"missingDataMethod":"IgnoreBlanks", "threshold":100.0, "weight":0,
"children":[{"type":"Child", "missingDataMethod":"IgnoreBlanks",
"threshold":80.0, "weight":0, "matchWhenNotTrue":false,
"scoringMethod":"Maximum",
"algorithms":[{"name":"EditDistance", "weight":0, "options":null},
{"name":"Metaphone", "weight":0, "options":null}],
"crossMatchField":[], "suspectField":"firstname", "candidateField":null},
{"type":"Child", "missingDataMethod":"IgnoreBlanks", "threshold":80.0,
 "weight":0,
"matchWhenNotTrue":false, "scoringMethod":"Maximum",
"algorithms":[{"name":"KeyboardDistance", "weight":0, "options":null},
{"name":"Metaphone3", "weight":0, "options":null}], "crossMatchField":[],
"suspectField":"lastname", "candidateField":null}],
"scoringMethod":"Average", "matchingMethod":"AllTrue", "name":"NameData",
 "matchWhenNotTrue":false};


-- Set the header for suspect table using configuration property
'pb.bdq.suspect.header'
set
pb.bdq.match.suspect.header=name,firstname,lastname,matchkey,middlename,recordid;

-- Set the header for candidate table using configuration property
'pb.bdq.candidate.header'
set
pb.bdq.match.candidate.header=name,firstname,lastname,matchkey,middlename,recordid;

-- Set the sorting field to the candidates unique id's alias used in
the query. This is not from the input data.
set pb.bdq.match.sort.field=c_id;

-- Set the express match column(optional)
```

```
set pb.bdq.match.express.column=matchkey;

-- Set sort field name to the alias used in the query, using
configuration property 'pb.bdq.match.inter.comparison'
set pb.bdq.match.inter.comparison=maxNumOfDuplicates,2;

-- Optionally, one can also set
'pb.bdq.match.inter.comparison=returnUniqueCandidates,true';

-- Set sort collection number option for unique records using
configuration property 'pb.bdq.match.unique.collectnumber.zero'
set pb.bdq.match.unique.collectnumber.zero=false;

-- Execute Query on the desired table. The query uses a UDF rowid, which
 must be present in the query to maintain the ordering of the data while
 reading.

SELECT lateralview.record ["MatchRecordType"],
 lateralview.record ["MatchScore"],
 lateralview.record ["HasDuplicate"],
 lateralview.record ["CollectionNumber"],
 coalesce(lateralview.record ["ExpressMatched"], ''),
 lateralview.record ["SourceType"],
 lateralview.record ["name"],
 lateralview.record ["firstname"],
 lateralview.record ["lastname"],
 lateralview.record ["matchkey"],
 lateralview.record ["middlename"],
 lateralview.record ["recordid"]
FROM (
 SELECT interMatch(s_id, s_name, s_firstname, s_lastname, s_matchkey,
s_middlename, s_recordid, c_id,c_name, c_firstname, c_lastname,
c_matchkey, c_middlename, c_recordid) AS
 OUTPUT
 FROM (
  SELECT suspects.suspect_id AS s_id,
   suspects.NAME AS s_name,
   suspects.firstname AS s_firstname,
   suspects.lastname AS s_lastname,
   suspects.matchkey AS s_matchkey,
   suspects.middlename AS s_middlename,
   suspects.recordid AS s_recordid,
   candidates.candidate_id AS c_id,
   candidates.NAME AS c_name,
   candidates.firstname AS c_firstname,
   candidates.lastname AS c_lastname,
   candidates.matchkey AS c_matchkey,
   candidates.middlename AS c_middlename,
   candidates.recordid AS c_recordid
  FROM

     (
   SELECT rowid(*) AS suspect_id
```

```
    ,*
   FROM namedataintersuspect
   ) AS suspects LEFT JOIN
   (
    SELECT rowid(*) AS candidate_id
    ,*
    FROM namedataintercandidate
   ) AS candidates
   on suspects.matchkey = candidates.matchkey

  ) AS joinrecords
 GROUP BY joinrecords.s_matchkey
 ) AS innerResult LATERAL VIEW explode(innerResult.OUTPUT) lateralview
 AS record;



-- Query to dump data to a file

INSERT OVERWRITE LOCAL DIRECTORY '/home/hadoop/intermatch/output'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
collection items terminated by '||' map keys terminated by ':'
SELECT lateralview.record ["MatchRecordType"],
 lateralview.record ["MatchScore"],
 lateralview.record ["HasDuplicate"],
 lateralview.record ["CollectionNumber"],
 coalesce(lateralview.record ["ExpressMatched"], ''),
 lateralview.record ["SourceType"],
 lateralview.record ["name"],
 lateralview.record ["firstname"],
 lateralview.record ["lastname"],
 lateralview.record ["matchkey"],
 lateralview.record ["middlename"],
 lateralview.record ["recordid"]
FROM (
 SELECT interMatch(s_id, s_name, s_firstname, s_lastname, s_matchkey,
s_middlename, s_recordid, c_id,c_name, c_firstname, c_lastname,
c_matchkey, c_middlename, c_recordid) AS
 OUTPUT
 FROM (
  SELECT suspects.suspect_id AS s_id,
   suspects.NAME AS s_name,
   suspects.firstname AS s_firstname,
   suspects.lastname AS s_lastname,
   suspects.matchkey AS s_matchkey,
   suspects.middlename AS s_middlename,
   suspects.recordid AS s_recordid,
   candidates.candidate_id AS c_id,
   candidates.NAME AS c_name,
   candidates.firstname AS c_firstname,
   candidates.lastname AS c_lastname,
   candidates.matchkey AS c_matchkey,
```

```
   candidates.middlename AS c_middlename,
   candidates.recordid AS c_recordid
  FROM

     (
   SELECT rowid(*) AS suspect_id
    ,*
   FROM namedataintersuspect
   ) AS suspects LEFT JOIN
   (
    SELECT rowid(*) AS candidate_id
    ,*
    FROM namedataintercandidate
   ) AS candidates
   on suspects.matchkey = candidates.matchkey

  ) AS joinrecords
 GROUP BY joinrecords.s_matchkey
 ) AS innerResult LATERAL VIEW explode(innerResult.OUTPUT) lateralview
 AS record;


-- Sample input Suspect data

--+----------------+---------+---------------+-----------+-----------+----------+
--| name             | firstname| lastname        | matchkey    |
middlename | recordid  |
--+----------------+---------+---------------+-----------+-----------+----------+
--| LAURA  ABADSANTOS| LAURA    | ABADSANTOS      | L           |
       | 1         |
--+----------------+---------+---------------+-----------+-----------+----------+

-- Sample input candidate data

--+----------------+---------+---------------+-----------+-----------+----------+
--| name             | firstname| lastname        | matchkey    |
middlename | recordid  |
--+----------------+---------+---------------+-----------+-----------+----------+
--| KATHRYN E ABATE  | KATHRYN  | ABATE           | L           |  E
       | 3         |
--| ANNA  ABAYEV     | ANNA     | ABAYEV          | L           |
       | 5         |
--+----------------+---------+---------------+-----------+-----------+----------+

-- Sample output data

-+--------+-----+------+---------+--------+-----+------+-----+-----+----+-----+------+
--|MatchRecordType|MatchScore|HasDuplicate|CollectionNumber|ExpressMatched|SourceType|
 name       | firstname| lastname|matchkey|middlename| recordid  |
-+--------+-----+------+---------+--------+-----+------+-----+-----+----+-----+------+
--|S              |0         |Y           |0-0-1           |
    |S          |LAURA ABADSA| LAURA    |ABADSANTO|    L   |          |
 1         |
```

```
--|D               |80         |D            |0-0-1            |N
    |C          |KATHRYN E AB| KATHRYN   |AB        |    L    |    E     |
 3          |
--|D               |90         |D            |0-0-1            |N
    |C          |ANNA ABAYEV | ANNA      |ABAYEV    |    L    |          |
 5          |
-+---------+------+-------+----------+--------+------+-------+-----+-----+-----+------+-------+
```

Intraflow Match

サンプル Hive スクリプト

```
-- Register Advance Matching Module[AMM] Hive UDF jar
ADD JAR <Directory path>/amm.hive.${project.version}.jar;

-- Provide alias to UDF class (optional). String in quotes represent
class names needed for this job to run.
CREATE TEMPORARY FUNCTION rowid as
'com.pb.bdq.hive.common.RowIDGeneratorUDF';

-- This rowid is needed by Intraflow Match to maintain the order of rows
 while creating groups. This is a UDF (User Defined Function) and
associates an incremental unique integer number to each row of the data.

CREATE TEMPORARY FUNCTION intraMatch as
'com.pb.bdq.amm.process.hive.intraflow.IntraMatchUDAF';
-- Intra Flow is implemented as a UDAF (User Defined Aggregation
function). It processes one group of rows at a time and generates the
result for that group of rows

-- Disable map side aggregation
set hive.map.aggr = false;

-- Set the rule using configuration property 'pb.bdq.match.rule'
set pb.bdq.match.rule={"type":"Parent",
"children":[{"type":"Child", "matchWhenNotTrue":false, "threshold":80.0,
 "weight":0,
"algorithms":[{"name":"EditDistance", "weight":0, "options":null},
{"name":"Metaphone", "weight":0, "options":null}],
"scoringMethod":"Maximum", "missingDataMethod":"IgnoreBlanks",
"crossMatchField":[], "suspectField":"firstname", "candidateField":null},
{"type":"Child", "matchWhenNotTrue":false, "threshold":80.0, "weight":0,
 "algorithms":[{"name":"KeyboardDistance", "weight":0, "options":null},
{"name":"Metaphone3", "weight":0, "options":null}],
"scoringMethod":"Maximum", "missingDataMethod":"IgnoreBlanks",
"crossMatchField":[], "suspectField":"lastname", "candidateField":null}],
"matchingMethod":"AllTrue", "scoringMethod":"Average",
```

```
"missingDataMethod":"IgnoreBlanks", "name":"NameData",
"matchWhenNotTrue":false, "threshold":100,"weight":0};


-- Set header(along with id field alias used in query) using
configuration property 'pb.bdq.match.header'
set pb.bdq.match.header=firstname,lastname,matchkey,middlename,id;

-- Set the express match column (optional)
set pb.bdq.match.express.column=matchkey;

-- Set sort field name to the alias used in the query, using the
configuration property 'pb.bdq.match.sort.field'
set pb.bdq.match.sort.field=id;

-- Set sort collection number option for unique records using
configuration property 'pb.bdq.match.unique.collectnumber.zero'
set pb.bdq.match.unique.collectnumber.zero=false;

-- Execute Query on the desired table. The query uses a UDF rowid, which
 must be present in the query to maintain the ordering of the data while
 reading.
-- Intra Match returns a list of map containing <key=value> pairs. Each
 map in the list corresponds to a row in the group. The below query
explodes that list of map and fetches fields from map by keys.

SELECT innerresult.record["MatchRecordType"],
 innerresult.record["MatchScore"],
 innerresult.record["CollectionNumber"],
 innerresult.record["ExpressMatched"],
 innerresult.record["firstname"],
 innerresult.record["lastname"],
 innerresult.record["matchkey"],
 innerresult.record["middlename"]
 FROM (
  SELECT  intraMatch(
    innerRowID.firstname,
    innerRowID.lastname,
    innerRowID.matchkey,
    innerRowID.middlename,
    innerRowID.id
  ) AS matchgroup
  FROM (
   SELECT  firstname, lastname, matchkey, middlename, rowid(*)
   AS id
   FROM customer_data
   ) innerRowID
  GROUP BY matchkey
 ) AS innerResult
 LATERAL VIEW explode(innerResult.matchgroup) innerresult AS record ;

-- Query to dump output to a file
```

```
INSERT OVERWRITE LOCAL DIRECTORY '/home/hadoop/IntraFlow/' ROW FORMAT
DELIMITED FIELDS TERMINATED BY ',' collection items terminated by '||'
 map keys terminated by ':'
SELECT innerresult.record["MatchRecordType"],
 innerresult.record["MatchScore"],
 innerresult.record["CollectionNumber"],
 innerresult.record["ExpressMatched"],
 innerresult.record["firstname"],
 innerresult.record["lastname"],
 innerresult.record["matchkey"],
 innerresult.record["middlename"]
 FROM (
  SELECT  intraMatch(innerRowID.firstname,
    innerRowID.lastname,
    innerRowID.matchkey,
    innerRowID.middlename,
    innerRowID.id
  ) AS matchgroup
  FROM (
    SELECT  firstname, lastname, matchkey, middlename, rowid(*)
    AS id
    FROM customer_data
   ) innerRowID
  GROUP BY matchkey
  ) AS innerResult
 LATERAL VIEW explode(innerResult.matchgroup) innerresult AS record ;


--sample input data
--+-------------+-------------+---------------+--------------+
--| firstname   | lastname    | middlename    | matchkey     |
--+-------------+-------------+---------------+--------------+
--| Steven      | Aaen        | LYRIC         | AAE          |
--| DEBRA       | AALMO       | BOATMAN       | AAE          |

--| MARY        | AARON       | ROLLING MEADOW| AAE          |
--+-------------+-------------+---------------+--------------+

--sample output data
—+————————+——————+————————+——————+———————————+———————————+———————————+————————+
--| firstname | lastname|middlename  |
matchkey|MatchRecordType|CollectionNumber|ExpressMatched|MatchScore|
—+————————+——————+————————+——————+———————————+———————————+———————————+————————+
--| Steven    | Aaen    | LYRIC      | AAE     | S        |
0-0-1  |     Y        |     0    |
--| DEBRA     | AALMO   | BOATMAN    | AAE     |    D         |
  0-0-1  |     Y        |    100   |
--| MARY      | AARON   | ROLLING MEA| AAE     |    D          |
   0-0-1  |     Y        |    100   |
```

```
—+————————+————————+————————+————————+————————————+————————————+————————————+————————+
```

Transactional Match

サンプル Hive スクリプト

```
-- Register Advance Matching Module[AMM] Hive UDF jar
ADD JAR <Directory path>/amm.hive.${project.version}.jar;

-- Provide alias to UDF class (optional). String in quotes represent
class names needed for this job to run.

CREATE TEMPORARY FUNCTION rowid as
'com.pb.bdq.hive.common.RowIDGeneratorUDF';

-- This rowid is needed by Transactional Match to maintain the order of
 rows while creating groups. This is a UDF (User Defined Function) and
 associates an incremental unique integer number to each row of the
data.

CREATE TEMPORARY FUNCTION transactionalMatch as
'com.pb.bdq.amm.process.hive.transactional.TransactionalMatchUDAF';

-- Transactional Match is implemented as a UDAF (User Defined Aggregation
 function). It processes one group of rows at a time and generates the
 result for that group of rows.

-- Disable map side aggregation
set hive.map.aggr = false;

-- Set the rule using configuration property 'pb.bdq.match.rule'
set pb.bdq.match.rule={"type":"Parent", "children":[{"type":"Child",
"matchWhenNotTrue":false, "threshold":80.0, "weight":0,
"algorithms":[{"name":"EditDistance", "weight":0, "options":null},
{"name":"Metaphone","weight":0,"options":null}],
"scoringMethod":"Maximum", "missingDataMethod":"IgnoreBlanks",
"crossMatchField":[], "suspectField":"firstname", "candidateField":null},
{"type":"Child", "matchWhenNotTrue":false, "threshold":80.0, "weight":0,
 "algorithms":[{"name":"KeyboardDistance","weight":0,"options":null},
{"name":"Metaphone3","weight":0,"options":null}],
"scoringMethod":"Maximum", "missingDataMethod":"IgnoreBlanks",
"crossMatchField":[], "suspectField":"lastname", "candidateField":null}],
 "matchingMethod":"AllTrue", "scoringMethod":"Average",
"missingDataMethod":"IgnoreBlanks", "name":"NameData",
"matchWhenNotTrue":false, "threshold":100, "weight":0};

-- Set header(along with id field alias used in query) using
```

```
configuration property 'pb.bdq.match.header'
set
pb.bdq.match.header=name,firstname,lastname,matchkey,middlename,recordid,id;

-- Set sort field name to the alias used in the query, using the
configuration property 'pb.bdq.match.sort.field'
set pb.bdq.match.sort.field=id;

-- Set sort collection number option for unique records using
configuration property 'pb.bdq.match.unique.candidate.return'. The
default value is false.
set pb.bdq.match.unique.candidate.return=true;

-- Execute Query on the desired table. The query uses a UDF rowid, which
 must be present in the query to maintain the ordering of the data while
 reading.
-- Transactional Match returns a list of map containing <key=value>
pairs. Each map in the list corresponds to a row in the group. The below
 query explodes that list of map and fetches fields from map by keys.

SELECT tmp2.record["MatchRecordType"],
 tmp2.record["MatchScore"],
 tmp2.record["HasDuplicate"],
 tmp2.record["name"],
 tmp2.record["firstname"],
 tmp2.record["lastname"],
 tmp2.record["matchkey"],
 tmp2.record["middlename"],
 tmp2.record["recordid"]
FROM (
 SELECT  transactionalMatch(innerRowID.name, innerRowID.firstname,
innerRowID.lastname, innerRowID.matchkey, innerRowID.middlename,
innerRowID.recordid, innerRowID.id
 ) AS matchgroup
 FROM (
  SELECT name, firstname, lastname, matchkey, middlename, recordid,
rowid(name, firstname, lastname, matchkey, middlename, recordid) AS id
 FROM customer_data
  ) innerRowID
 GROUP BY matchkey
) As innerResult
LATERAL VIEW explode(innerResult.matchgroup) tmp2 as record ;


-- Query to dump output to a file

INSERT OVERWRITE LOCAL DIRECTORY '/home/hadoop/transmatch/' ROW FORMAT
 DELIMITED FIELDS TERMINATED BY ',' collection items terminated by '||'
 map keys terminated by ':'
SELECT tmp2.record["MatchRecordType"],
 tmp2.record["MatchScore"],
 tmp2.record["HasDuplicate"],
 tmp2.record["name"],
```

```
 tmp2.record["firstname"],
 tmp2.record["lastname"],
 tmp2.record["matchkey"],
 tmp2.record["middlename"],
 tmp2.record["recordid"]
FROM (
 SELECT  transactionalMatch(innerRowID.name,
   innerRowID.firstname,
   innerRowID.lastname,
   innerRowID.matchkey,
   innerRowID.middlename,
   innerRowID.recordid,
   innerRowID.id) as matchgroup
 FROM (
  SELECT name, firstname, lastname, matchkey, middlename, recordid,
rowid(name, firstname, lastname, matchkey, middlename, recordid) AS id

  FROM customer_data
  ) innerRowID
 GROUP BY matchkey ) As innerResult
LATERAL VIEW explode(innerResult.matchgroup) tmp2 as record ;



--sample input data
-+----------+-------------+-------------+-------------+-------------+-+
--| name               | firstname               | lastname
 | matchkey              | middlename              | recordid
     |
-+----------+-------------+-------------+-------------+-------------+-+
--| ZORINA  ABDOOL     | ZORINA                  | ABDOOL
 | Z                     |                         | 12
     |
--| ZULFIQAR  ALI      | ZULFIQAR                | ALI
 | Z                     |                         | 116
     |
--| ZACHARY  BENNETT   | ZACHARY                 | BENNETT
 | Z                     |                         | 515
     |
--| ZOHAR  BUERGER     | ZOHAR                   | BUERGER
 | Z                     |                         | 889
     |
-+----------+-------------+-------------+-------------+-------------+-+

--sample output data
-+----------+-------------+-------+-------+-------+-----------+-------+--------+
--|name            |firstname | lastname | matchkey | middlename |
recordid  | MatchRecordType | MatchScore | HasDuplicate |
-+----------+-------+-------+-------+--------+-------+-----------+-------+--------+
--|ZORINA  ABDOOL  |ZORINA    | ABDOOL   | Z        |           | 12
       |  S     | 0  |  Y     |
--|ZULFIQAR  ALI   |ZULFIQAR  | ALI      | Z        |           | 116
       |      D          | 90  |  D    |
```

```
--|ZACHARY   BENNETT|ZACHARY    | BENNETT   | Z         |             | 515
        |       D           | 91   |   D     |
--|ZOHAR   BUERGER   |ZOHAR      | BUERGER   | Z         |             | 889
        |   D      | 91   |   D     |
--+------------+-------+---------------+-----------+---------+----------+
```

Best of Breed

サンプル Hive スクリプト

```
-- Register Advance Matching Module[AMM] Hive UDF jar
ADD JAR <Directory path>/amm.hive.${project.version}.jar;

-- Provide alias to UDF class (optional). String in quotes represent
class names needed for this job to run.

CREATE TEMPORARY FUNCTION rowid as
'com.pb.bdq.hive.common.RowIDGeneratorUDF';

-- This rowid is needed by Best of Breed to maintain the order of rows
 while creating groups. This is a UDF (User Defined Function) and
associates an incremental unique integer number to each row of the data.

CREATE TEMPORARY FUNCTION bestofbreed as
'com.pb.bdq.amm.process.hive.consolidation.bestofbreed.BestOfBreedUDAF';
-- Best of Breed is implemented as a UDAF (User Defined Aggregation
function). It processes one group of rows at a time and generates the
result for that group of rows.

-- Disable map side aggregation
set hive.map.aggr = false;

-- Set the rule using configuration property 'pb.bdq.consolidation.rule'

set pb.bdq.consolidation.rule={"consolidationConditions":[
{"consolidationRule":{"conditionClass":"conjoinedRule", "joinType":"AND",
"consolidationRules":[{"conditionClass":"simpleRule",
"operation":"LONGEST", "fieldName":"c5", "value":null,
"valueNumeric":true, "valueFromField":false},
{"conditionClass":"simpleRule", "operation":"IS_NOT_EMPTY",
"fieldName":"c9", "value":null, "valueNumeric":false,
"valueFromField":false}]},
"actions":[{"accumulate":false, "copyFromField":true, "sourceData":"c2",
 "destinationFieldName":"c2"},
{"accumulate":false, "copyFromField":false, "sourceData":"Admin",
"destinationFieldName":"c4"}]},
{"consolidationRule":{"conditionClass":"conjoinedRule", "joinType":"AND",
```

```
"consolidationRules":[{"conditionClass":"simpleRule",
"operation":"LONGEST", "fieldName":"c5", "value":null,
"valueNumeric":true, "valueFromField":false},
{"conditionClass":"simpleRule", "operation":"IS_NOT_EMPTY",
"fieldName":"c9", "value":null, "valueNumeric":false,
"valueFromField":false}]},
"actions":[{"accumulate":false, "copyFromField":false,
"sourceData":"Changed", "destinationFieldName":"c10"},
{"accumulate":false, "copyFromField":true, "sourceData":"c5",
"destinationFieldName":"c6"},
{"accumulate":true, "copyFromField":true, "sourceData":"c10",
"destinationFieldName":"c10"}]}],
"keepOriginalRecords":true, "buildTemplateRecord":true,
"templateRules":[{"consolidationRule":{"conditionClass":"conjoinedRule",
 "joinType":"OR",
"consolidationRules":[{"conditionClass":"simpleRule",
"operation":"CONTAINS", "fieldName":"c1", "value":"li",
"valueNumeric":false, "valueFromField":false},
{"conditionClass":"simpleRule", "operation":"LONGEST", "fieldName":"c5",
 "value":null, "valueNumeric":false, "valueFromField":false}]},
"actions":[]}]};

-- Set header (along with the id field alias used in the query) using
configuration property 'pb.bdq.consolidation.header'
set pb.bdq.consolidation.header=c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,id;

-- Set sort field name to the alias used in the query, using the
configuration property 'pb.bdq.consolidation.sort.field'
set pb.bdq.consolidation.sort.field=id;

-- Execute Query on the desired table. The query uses a UDF rowid, which
 must be present in the query to maintain the ordering of the data while
 reading.
-- Best of Breed returns a list of map containing <key=value> pairs.
Each map in the list corresponds to a row in the group. The below query
 explodes that list of map and fetches fields from map by keys.

SELECT tmp2.record["c1"],
 tmp2.record["c2"],
 tmp2.record["c3"],
 tmp2.record["c4"],
 tmp2.record["c5"],
 tmp2.record["c6"],
 tmp2.record["c7"],
 tmp2.record["c8"],
 tmp2.record["c9"],
 tmp2.record["c10"],
 tmp2.record["CollectionRecordType"]
FROM (
 SELECT  bestofbreed(innerRowID.c1,
  innerRowID.c2,
  innerRowID.c3,
  innerRowID.c4,
```

```
  innerRowID.c5,
  innerRowID.c6,
  innerRowID.c7,
  innerRowID.c8,
  innerRowID.c9,
  innerRowID.c10,
  innerRowID.id) AS matchgroup
 FROM(
  SELECT c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, rowid(*) AS id FROM
databob
  ) innerRowID
 GROUP BY c3
 ) AS innerResult
LATERAL VIEW explode(innerResult.matchgroup) tmp2 AS record ;

-- Query to dump the output to a file

INSERT OVERWRITE LOCAL DIRECTORY '/home/hadoop/bestofbreed/' ROW FORMAT
 DELIMITED FIELDS TERMINATED BY ',' collection items terminated by '||'
 map keys terminated by ':'
SELECT tmp2.record["c1"],
 tmp2.record["c2"],
 tmp2.record["c3"],
 tmp2.record["c4"],
 tmp2.record["c5"],
 tmp2.record["c6"],
 tmp2.record["c7"],
 tmp2.record["c8"],
 tmp2.record["c9"],
 tmp2.record["c10"],
 tmp2.record["CollectionRecordType"]
FROM (
 SELECT bestofbreed(innerRowID.c1,
  innerRowID.c2,
  innerRowID.c3,
  innerRowID.c4,
  innerRowID.c5,
  innerRowID.c6,
  innerRowID.c7,
  innerRowID.c8,
  innerRowID.c9,
  innerRowID.c10,
  innerRowID.id) as matchgroup
 FROM(
  SELECT c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, rowid(*) AS id FROM
databob
  ) innerRowID
 GROUP BY c3
 ) AS innerResult
LATERAL VIEW explode(innerResult.matchgroup) tmp2 AS record ;


--sample input data
```

```
-+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
--| c1       |      c2  |      c3  |      c4  |      c5  |      c6  |
     c7  |      c8  | c9       | c10      |
-+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
--| Duplicate| 87       | 1        |          |ANNA ABNEY| ANNA     |
         | ABNEY    | A        | 18       |
--| Duplicate| 77       | 1        |          |ANNA A ANN| ANDREA   |
         | ANNAKAY  | A        | 196      |
-+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+

--sample output data
-+------+------+------+------+------+------+------+------+------+-------------+
--| c1       | c2       | c3       | c4       | c5       | c6       |
c7       | c8       | c9       | c10      |CollectionRecordType|
-+------+------+------+------+------+------+------+------+------+-------------+
--| Duplicate| 87       | 1        |          |ANNA ABNEY| ANNA     |
         | ABNEY    | A        | 18       |   Primary          |
--| Duplicate| 77       | 1        |          |ANNA A ANN| ANDREA   |
 ARANOW  | ANNAKAY  | A        | 196      |   Secondary        |
--| Duplicate| 87       | 1        |          |ANNA ABNEY| ANNA     |
 ARANOW  | ABNEY    | A        | 18       |   BestOfBreed      |
-+------+------+------+------+------+------+------+------+------+-------------+
```

Duplicate Synchronization

サンプル Hive スクリプト

```
-- Register Advance Matching Module[AMM] Hive UDF jar
ADD JAR <Directory path>/amm.hive.${project.version}.jar;

-- Provide alias to UDF class (optional). String in quotes represent
class names needed for this job to run.

CREATE TEMPORARY FUNCTION rowid as
'com.pb.bdq.hive.common.RowIDGeneratorUDF';

-- This rowid is needed by Duplicate Synchronization to maintain the
order of rows while creating groups. This is a UDF (User Defined
Function) and associates an incremental unique integer number to each
row of the data.

CREATE TEMPORARY FUNCTION dupsync as
'com.pb.bdq.amm.process.hive.consolidation.duplicatesync.DuplicateSyncUDAF';

-- Duplicate Sync is implemented as a UDAF (User Defined Aggregation
function). It processes one group of rows at a time and generates the
result for that group of rows.
```

```
-- Disable map side aggregation
set hive.map.aggr = false;

-- Set the rule using configuration property 'pb.bdq.consolidation.rule'

set pb.bdq.consolidation.rule={"consolidationConditions":
[{"consolidationRule":
{"conditionClass":"conjoinedRule", "joinType":"AND",
"consolidationRules":[{"conditionClass":"simpleRule",
"operation":"HIGHEST", "fieldName":"column2", "value":null,
"valueFromField":false, "valueNumeric":true}]},
"actions":[{"accumulate":false, "copyFromField":true,
"sourceData":"column5", "destinationFieldName":"column5"}]}]};

-- Set header (along with the id field alias used in the query) using
configuration property 'pb.bdq.consolidation.header'
set
pb.bdq.consolidation.header=column1,column2,column3,column4,column5,id;

-- Set sort field name to alias used in query using configuration
property 'pb.bdq.consolidation.sort.field'
set pb.bdq.consolidation.sort.field=id;

-- Execute Query on the desired table. The query uses a UDF rowid, which
 must be present in the query to maintain the ordering of the data while
 reading.
-- Duplicate Sync returns a list of map containing <key=value> pairs.
Each map in the list corresponds to a row in the group. The below query
 explodes that list of map and fetches fields from map by keys.

SELECT tmp2.record["column1"],
 tmp2.record["column2"],
 tmp2.record["column3"],
 tmp2.record["column4"],
 tmp2.record["column5"]
FROM (
 SELECT  dupsync (innerRowID.column1,
  innerRowID.column2,
  innerRowID.column3,
  innerRowID.column4,
  innerRowID.column5,
  innerRowID.id
  ) AS matchgroup
 FROM (
  SELECT column1, column2, column3, column4, column5, rowid(*)
  AS id
  FROM databob
  ) innerRowID
 GROUP BY column3
 ) AS innerResult
 LATERAL VIEW explode(innerResult.matchgroup) tmp2 AS record ;
```

```
-- Query to dump the output to a file

INSERT OVERWRITE LOCAL DIRECTORY '/home/hadoop/dupsync/' ROW FORMAT
DELIMITED FIELDS TERMINATED BY ',' collection items terminated by '||'
 map keys terminated by ':'
SELECT tmp2.record["column1"],
 tmp2.record["column2"],
 tmp2.record["column3"],
 tmp2.record["column4"],
 tmp2.record["column5"]
FROM (
 SELECT dupsync( innerRowID.column1,
  innerRowID.column2,
  innerRowID.column3,
  innerRowID.column4,
  innerRowID.column5,
  innerRowID.id
  ) AS matchgroup
 FROM (
  SELECT column1, column2, column3, column4, column5, rowid(*)
  AS id
  FROM databob
  ) innerRowID
 GROUP BY column3 ) AS innerResult
 LATERAL VIEW explode(innerResult.matchgroup) tmp2 AS record ;



--sample input data
--+----------+----------+----------+----------+----------+
--| column1  | column2  | column3  | column4  | column5  |
--+----------+----------+----------+----------+----------+
--| Duplicate| 87       | 1        |          |ANNA ABNEY|
--| Duplicate| 77       | 1        |          |ANNA A ANN|
--| Suspect  |          | 1        |          |ANNA A ABN|
--+----------+----------+----------+----------+----------+

--sample output data
--+----------+----------+----------+----------+----------+
--| column1  | column2  | column3  | column4  | column5  |
--+----------+----------+----------+----------+----------+
--| Duplicate| 87       | 1        |          |ANNA ABNEY|
--| Duplicate| 77       | 1        |          |ANNA A ANN|
```

```
--| Suspect  |          | 1        |          |ANNA ABNEY|
--+----------+----------+----------+----------+----------+
```

フィルタ

サンプル Hive スクリプト

```
-- Register Advance Matching Module[AMM] Hive UDF jar
ADD JAR <Directory path>/amm.hive.${project.version}.jar;

-- Provide alias to UDF class (optional). String in quotes represent
class names needed for this job to run.

CREATE TEMPORARY FUNCTION rowid as
'com.pb.bdq.hive.common.RowIDGeneratorUDF';

-- This rowid is needed by Filter to maintain the order of rows while
creating groups. This is a UDF (User Defined Function) and associates
an incremental unique integer number to each row of the data.

CREATE TEMPORARY FUNCTION filter as
'com.pb.bdq.amm.process.hive.consolidation.filter.FilterUDAF';

-- Filter is implemented as a UDAF (User Defined Aggregation function).
 It processes one group of rows at a time and generates the result for
 that group of rows.

-- Disable map side aggregation
set hive.map.aggr = false;

-- Set the rule using configuration property 'pb.bdq.consolidation.rule'
set pb.bdq.consolidation.rule={"consolidationConditions":
[{"consolidationRule":{"conditionClass":"simpleRule",
"operation":"HIGHEST", "fieldName":"column2", "value":null,
"valueFromField":false, "valueNumeric":true}, "actions":[]}],
"removeDuplicates":true};

-- Set header (along with the id field alias used in the query) using
configuration property 'pb.bdq.consolidation.header'
set
pb.bdq.consolidation.header=column1,column2,column3,column4,column5,id;

-- Set sort field name to alias used in query using configuration
property 'pb.bdq.consolidation.sort.field'
set pb.bdq.consolidation.sort.field=id;

-- Execute Query on the desired table. The query uses a UDF rowid, which
 must be present in the query to maintain the ordering of the data while
```

```
 reading.

SELECT tmp2.record["column1"],
 tmp2.record["column2"],
 tmp2.record["column3"],
 tmp2.record["column4"],
 tmp2.record["column5"]
FROM (
 SELECT  filter (innerRowID.column1,
   innerRowID.column2,
   innerRowID.column3,
   innerRowID.column4,
   innerRowID.column5,
   innerRowID.id
 ) AS matchgroup
 FROM (
  SELECT column1, column2, column3, column4, column5, rowid(*)
  AS id
  FROM data
  ) innerRowID
 GROUP BY column3
 ) AS innerResult
LATERAL VIEW explode(innerResult.matchgroup) tmp2 AS record ;

-- Query to dump the output to a file

INSERT OVERWRITE LOCAL DIRECTORY '/home/hadoop/HiveUDF/filter/'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
collection items terminated by '||' map keys terminated by ':'
SELECT tmp2.record["column1"],
 tmp2.record["column2"],
 tmp2.record["column3"],
 tmp2.record["column4"],
 tmp2.record["column5"]
FROM (
 SELECT  filter (innerRowID.column1,
   innerRowID.column2,
   innerRowID.column3,
   innerRowID.column4,
   innerRowID.column5,
   innerRowID.id
 ) AS matchgroup
 FROM (
  SELECT column1, column2, column3, column4, column5, rowid(*)
  AS id
  FROM data
  ) innerRowID
 GROUP BY column3
 ) AS innerResult
LATERAL VIEW explode(innerResult.matchgroup) tmp2 AS record ;
```

```
--sample input data
--+----------+----------+----------+----------+----------+
--| column1  | column2  | column3  | column4  | column5  |
--+----------+----------+----------+----------+----------+
--| Duplicate| 80       | 98       |          | EUNICE L |
--| Suspect  |          | 98       |          | ERIC L BR|
--+----------+----------+----------+----------+----------+

--sample output data
--+----------+----------+----------+----------+----------+
--| column1  | column2  | column3  | column4  | column5  |
--+----------+----------+----------+----------+----------+
--| Suspect  |          | 98       |          | ERIC L BR|
--+----------+----------+----------+----------+----------+
```

Data Normalization モジュールの関数

Table Lookup

サンプル Hive スクリプト

```
-- Register Data Normalization Modue [dnm] BDQ Hive UDF Jar
ADD JAR <Directory path>/dnm.hive.${project.version}.jar;

-- Provide alias to UDF class (optional). String in quotes represent
class names needed for this job to run.
-- Table Lookup is implemented as a UDF (User Defined function). Hence
 it processes one row at a time and generates a map of key value pairs
 for each row.
CREATE TEMPORARY FUNCTION tablelookup as
'com.pb.bdq.dnm.process.hive.tablelookup.TableLookUpUDF';

-- Set rule
set rule='{"rules":[{"action":"Standardize", "source":"CityCode",
"tableName":"State Name Abbreviations", "lookupMultipleWordTerms":false,
 "lookupIndividualTermsWithinField":false, "destination":"CityCode"}]}';

-- Set Reference Directory. This must be a local path on cluster machines
 and must be present on each node of the cluster at the same path.
set refdir='/home/hadoop/reference';

-- set header
```

```
set header ='AccountDescription,Address,ApartmentNumber,CityCode';


-- Execute Query on the desired table, to display the job output on
console. This query returns a map of key value pairs containing output
 fields for each row.

SELECT bar.ret["StandardizationTermIdentified"],
 bar.ret["accountdescription"],
 bar.ret["address"],
 bar.ret["apartmentnumber"],
 bar.ret["citycode"]
FROM (
 SELECT tablelookup(${hiveconf:rule}, ${hiveconf:refdir},
${hiveconf:header}, accountdescription, address, apartmentnumber,
citycode)
 AS ret
 FROM citizen_data
 ) bar;


-- Query to dump output data to a file

INSERT OVERWRITE LOCAL DIRECTORY '/home/hadoop/TableLookup/' row format
 delimited FIELDS TERMINATED BY ',' lines terminated by '\n' STORED AS
 TEXTFILE
SELECT bar.ret["StandardizationTermIdentified"],
 bar.ret["accountdescription"],
 bar.ret["address"],
 bar.ret["apartmentnumber"],
 bar.ret["citycode"]
FROM (
 SELECT tablelookup(${hiveconf:rule}, ${hiveconf:refdir},
${hiveconf:header}, accountdescription, address, apartmentnumber,
citycode)
 AS ret
 FROM citizen_data
 ) bar;

--Sample input data
--+-------------------------+----------------+----------------------+-----------------+
--| citizen_data.accountdescription  | citizen_data.address  |
citizen_data.apartmentnumber  | citizen_data.citycode |
--+-------------------------+----------------+----------------------+-----------------+
--|                          | 400 E M0 St Apt 1405  |
            | NY                        |
--|                                     | 190 E 72nd St         |
                        | NY                          |
--|                                     | 1381 3rd Ave Apt 4    | 4
                        | TTYYY                       |
--+-------------------------+----------------+----------------------+-----------------+
```

```
--sample output data
--+---------------------+-----------------+----------------+----------------+-------+
--|StandardizationTermIdentified |  accountdescription   | address
  | apartmentnumber    | citycode|
--+---------------------+-----------------+----------------+----------------+-------+

--|    yes     |              | 400 E M0 St Apt 1405  |
       | NEW YORK |
--|    yes          |                            | 190 E 72nd St
|                        | NEW YORK |
--|    yes     |                           | 1381 3rd Ave Apt 4    | 4
                     | NEW YORK |
--+---------------------+-----------------+----------------+----------------+-------+
```

Advanced Transformer

サンプル Hive スクリプト

```
-- Register Data Normalisation Module [DNM] BDQ Hive UDF Jar
ADD JAR <Directory path>/dnm.hive.${project.version}.jar;

-- Provide alias to UDF class (optional). String in quotes represent
class names needed for this job to run.
-- Advanced Transformer is implemented as a UDF (User Defined function).
 Hence it processes one row at a time and generates a map of key value
 pairs for each row.
CREATE TEMPORARY FUNCTION advanceTransform as
'com.pb.bdq.dnm.process.hive.advancetransformer.AdvanceTransformerUDF';

-- Set rule
set rule='{"rules":[{"extractionType":"TableData", "source":"address",
 "nonExtractedData":"address_1", "extractedData":"address_2",
"tokenizationCharacters":"", "tableName":"Street Suffix Abbreviations",
 "multipleTermLookup":false, "tokenize":true, "extract":"ExtractTerm",
 "includeTermWith":"ExtractedData", "wordsToExtract":2}]}';


-- Set Reference Directory. This must be a local path on cluster machines
 and must be present on each node of the cluster at the same path.
set refdir='/home/hadoop/reference/';

-- set header
set header ='AccountDescription,Address';

-- Execute Query on the desired table, to display the job output on
console. This query returns a map of key value pairs containing output
```

```
 fields for each row.

SELECT bar.ret["AdvancedTransformTermIdentified"],
 bar.ret["accountdescription"],
 bar.ret["address"],
 bar.ret["address_1"]
FROM (
 SELECT advanceTransform(${hiveconf:rule}, ${hiveconf:refdir},
${hiveconf:header}, accountdescription, address)
 AS ret
 FROM advxformX
 ) bar;


-- Query to dump output data to a file

INSERT OVERWRITE LOCAL DIRECTORY '/home/hadoop/AdvXformer/' row format
 delimited FIELDS TERMINATED BY ',' lines terminated by '\n' STORED AS
 TEXTFILE
SELECT bar.ret["AdvancedTransformTermIdentified"],
 bar.ret["accountdescription"],
 bar.ret["address"],
 bar.ret["address_1"]
FROM (
 SELECT advanceTransform(${hiveconf:rule}, ${hiveconf:refdir},
${hiveconf:header}, accountdescription, address)
 AS ret
 FROM advxformX
 ) bar;

--sample input data
+---------------------------------+--------------------+----------------------+
| AdvancedTransformTermIdentified  | accountdescription  |      address
          |
+---------------------------------+--------------------+----------------------+
| Yes                              |           | 400 E M0 St Apt 1405
 |
| Yes                              |                    | 190 E 72nd
St         |
+---------------------------------+--------------------+----------------------+

--sample output data
+---------------------------+-----------------+-------------------+-----------------+
| AdvancedTransformTermIdentified  | accountdescription  |      address
          |     address_1      |
+---------------------------+-----------------+-------------------+-----------------+
| Yes                              |           | 400 E M0 St Apt 1405
 | 400 E M0 Apt 1405  |
| Yes                              |                    | 190 E 72nd
St         | 190 E 72nd         |
```

```
+-----------------------------+-----------------+-----------------+-----------------+
```

Universal Addressing モジュールの機能

Validate Address

重要： 最初の Validate Address ジョブを作成および実行する前に、Acushare サービスが実行されていることを確認します。手順については、**Acushare サービスの実行**（12ページ）を参照してください。

サンプル Hive スクリプト

```
-- Register Universal Addressing Module [UAM-Global] BDQ Hive UDAF Jar

ADD JAR <Directory
path>/uam.universaladdress.hive.${project.version}.jar;

-- Provide alias to UDAF class (optional). String in quotes represent
class names needed for this job to run.
CREATE TEMPORARY FUNCTION uamvalidation as
'com.pb.bdq.uam.process.hive.universaladdress.UAMUSAddressingUDAF';

-- set LD_LIBRARY_PATH(path to modules lib, runtime/lib and runtime/bin),
 G1RTS(path containing COBOL runtime) and ACU_RUNCBL_JNI_ONLOAD_DISABLE
 in this configuration
set mapreduce.admin.user.env =
LD_LIBRARY_PATH=/home/hduser/~/runtime/lib:
/home/hduser/~/runtime/bin:/home/hduser/~/server/modules/universaladdress/lib,
 ACU_RUNCBL_JNI_ONLOAD_DISABLE=1, G1RTS=/home/hduser/~/ ;

set hive.map.aggr = false;

-- set engine configuration
set pb.bdq.uam.universaladdress.engine.configurations={ "referenceData":{

"dataDir":"/home/hduser/resources/uam/universaladdress/UAM_universaladdress4.0_Feb15/",
 "referenceDataPathLocation":"LocaltoDataNodes"},
"cobolRuntimePath":"/home/hduser/tapan/addressquality/",
"modulesDir":"/home/hduser/tapan/addressquality/modules",
"dpvDbPath":null, "suiteLinkDBPath":null, "ewsDBPath":null,
"rdiDBPath":null, "lacsDBPath":null};
```

```
-- set input configuration
set
pb.bdq.uam.universaladdress.input.configuration={"outputStandardAddress":true,
 "outputPostalData":false, "outputParsedInput":false,
"outputAddressBlocks":true, "performUSProcessing":true,
"performCanadianProcessing":false,
"performInternationalProcessing":false, "outputFormattedOnFail":false,
 "outputCasing":"MIXED", "outputPostalCodeSeparator":true,
"outputMultinationalCharacters":false, "performDPV":false,
"performRDI":false, "performESM":false, "performASM":false,
"performEWS":false, "performLACSLink":false, "performLOT":false,
"failOnCMRAMatch":false, "extractFirm":false, "extractUrb":false,
"outputReport3553":false, "outputReportSERP":false,
"outputReportSummary":true, "outputCASSDetail":false,
"outputFieldLevelReturnCodes":false, "keepMultimatch":false,
"maximumResults":10,
"standardAddressFormat":"STANDARD_ADDRESS_FORMAT_COMBINED_UNIT",
"standardAddressPMBLine":"STANDARD_ADDRESS_PMB_LINE_NONE",
"cityNameFormat":"CITY_FORMAT_STANDARD", "vanityCityFormatLong":true,
"outputCountryFormat":"ENGLISH", "homeCountry":"United States",
"streetMatchingStrictness":"MATCHING_STRICTNESS_MEDIUM",
"firmMatchingStrictness":"MATCHING_STRICTNESS_MEDIUM",
"directionalMatchingStrictness":"MATCHING_STRICTNESS_MEDIUM",
"dualAddressLogic":"DUAL_NORMAL", "dpvSuccessfulStatusCondition":"A",
"reportListFileName":"", "reportlistProcessorName":"",
"reportlistNumber":1, "reportMailerAddress":"", "reportMailerName":"",
 "reportMailerCityLine":"", "canReportMailerCPCNumber":"",
"canReportMailerAddress":"", "canReportMailerName":"",
"canReportMailerCityLine":"", "internationalCityStreetSearching":100,
"addressLineSearchOnFail":true, "outputStreetAlias":true,
"outputVeriMoveBlock":false, "dpvDetermineNoStat":false,
"dpvDetermineVacancy":false, "outputAbbreviatedAlias":false,
"outputPreferredAlias":false,
"outputPreferredCity":"CITY_OVERRIDE_NAME_ZIP4",
"performSuiteLink":false, "suppressZplusPhantomCarrierR777":false,
"canStandardAddressFormat":"D", "canEnglishApartmentLabel":"APT",
"canFrenchApartmentLabel":"APP", "canFrenchFormat":"C",
"canOutputCityFormat":"D", "canOutputCityAlias":true,
"canDualAddressLogic":"D", "canPreferHouseNum":false,
"canSSLVRFLG":false, "canRuralRouteFormat":"A", "canNonCivicFormat":"A",
 "canDeliveryOfficeFormat":"I", "canEnableSERP":false,
"canSwitchManagedPostalCodeConfidence":false, "stats":null,
"counts":null, "z3seg":null, "serpStats":null, "dpvSeedList":null,
"lacsSeedList":null, "zipInputSet":null, "reportName":null,
"currentUser":null, "jobName":null, "jobId":null, "jobRequest":false,
"properties":{"DPVDetermineVacancy":"N", "DualAddressLogic":"N",
"ExtractUrb":"N", "CanFrenchFormat":"C", "AddressLineSearchOnFail":"Y",
 "OutputFieldLevelReturnCodes":"N", "OutputFormattedOnFail":"N",
"OutputStreetNameAlias":"Y", "OutputReportSERP":"N",
"OutputAddressBlocks":"Y", "ExtractFirm":"N",
"CanEnglishApartmentLabel":"APT", "OutputPreferredCity":"Z",
"FirmMatchingStrictness":"M", "CanFrenchApartmentLabel":"APP",
```

```
"KeepMultimatch":"N", "StandardAddressPMBLine":"N",
"PerformSuiteLink":"N", "CanStandardAddressFormat":"D",
"DPVSuccessfulStatusCondition":"A", "PerformLACSLink":"N",
"PerformUSProcessing":"Y", "PerformEWS":"N", "StandardAddressFormat":"C",
 "SuppressZplusPhantomCarrierR777":"N", "HomeCountry":"United States",
 "ReportMailerAddress":"", "OutputReport3553":"N",
"OutputVeriMoveDataBlock":"N", "CanDeliveryOfficeFormat":"I",
"OutputAbbreviatedAlias":"N", "PerformCanadianProcessing":"N",
"PerformDPV":"N", "PerformInternationalProcessing":"N",
"CanSSLVRFlg":"N", "StreetMatchingStrictness":"M",
"InternationalCityStreetSearching":"100",
"canSwitchManagedPostalCodeConfidence":"N", "CanDualAddressLogic":"D",
 "PerformASM":"N", "OutputCasing":"M", "ReportListFileName":"",
"CanReportMailerAddress":"", "ReportMailerCityLine":"",
"CanReportMailerCPCNumber":"", "ReportListProcessorName":"",
"CanOutputCityAlias":"Y", "DirectionalMatchingStrictness":"M",
"CanRuralRouteFormat":"A", "CanOutputCityFormat":"D",
"ReportListNumber":"1", "CanReportMailerCityLine":"",
"OutputMultinationalCharacters":"N", "EnableSERP":"N",
"CanNonCivicFormat":"A", "OutputShortCityName":"S",
"OutputPostalCodeSeparator":"Y", "FailOnCMRAMatch":"N", "PerformLOT":"N",
 "OutputCountryFormat":"E", "CanPreferHouseNum":"N",
"CanReportMailerName":"", "PerformRDI":"N", "ReportMailerName":"",
"PerformESM":"N", "OutputReportSummary":"Y",
"OutputVanityCityFormatLong":"Y", "OutputPreferredAlias":"N",
"DPVDetermineNoStat":"N", "MaximumResults":"10"}}};

-- set general configuration
set pb.bdq.uam.universaladdress.general.configuration =
{"dFileType":"SPLIT", "dMemoryModel":"MEDIUM",
"lacsLinkMemoryModel":"MEDIUM", "suiteLinkMemoryModel":"MEDIUM"};

-- set reference path
set pb.bdq.reference.data.local.location=/media/New
Volume/hduser/resources/uam/universaladdress/UAM_universaladdress4.0_Feb15;

-- set process type
set pb.bdq.uam.universaladdress.process.type=VALIDATE;

-- set header
set pb.bdq.header=InputKeyValue,FirmName,AddressLine1,AddressLine2,City,
StateProvince,PostalCode,Text;

-- Execute Query on the desired table, to display the job output on
console. This query returns a map of key value pairs containing output
 fields for each row.
SELECT tmp2.record["Confidence"], tmp2.record["AddressLine1"] FROM (
select uamvalidation(inputkeyvalue, firmname, addressline1, addressline2,
 city, stateprovince, postalcode, text) from uam_us) as addressgroup
LATERAL VIEW explode(addressgroup.mygp) tmp2 as record ;

-- Query to dump output data to a file
INSERT OVERWRITE LOCAL DIRECTORY '/home/hadoop/GlobalAddressing/' row
```

```
format delimited FIELDS TERMINATED BY ',' lines terminated by '\n' STORED
 AS TEXTFILE
SELECT tmp2.record["Confidence"], tmp2.record["AddressLine1"] FROM (
select uamvalidation(inputkeyvalue, firmname, addressline1, addressline2,
 city, stateprovince, postalcode, text) from uam_us) as addressgroup
LATERAL VIEW explode(addressgroup.mygp) tmp2 as record ;
```

```
+-----------+-------------------+----------+---------------+-------------+-----------+
| address.recordid  |     address.addressline1     |  address.city   |
 address.stateprovince  | address.postalcode  | address.country  |
+-----------+-------------------+----------+---------------+-------------+-----------+
| 1                 | 18 Merivale St               | South Brisbane  |
 QLD                    | 4101                | AUS              |
| 2                 | 19 Serpentine Rd             | Albany          |
 WA                     | 6330                | AUS              |
| 3                 | 317 VICTORIA ST GR           | BRUNSWICK       |
 VIC                    | 3056                | AUS              |
| 4                 | DUPLEX 6/16-18 O'CONNELL ST  | AINSLIE         |
 ACT                    | 2602                | AUS              |
| 5                 | LOT 154 470 BRYGON CREEK DR  | UPPER COOMERA   |
 QLD                    | 4209                | AUS              |
| 6                 | 16 GREENE ST                 | WARRAWONG       |
 ACT                    | 2502                | AUS              |
| 7                 | UNIT 47/16 BLAIRMOUNT ST     | PARKINSON       |
 QLD                    | 4115                | AUS              |
| 8                 | 13-15 FRANCESCO CRES         | BELLA VISTA     |
 NSW                    | 2153                | AUS              |
| 9                 | 4 RYANS LANE                 | HEATHCOTE       |
 VIC                    | 3523                | AUS              |
| 10                | 1 CHRISTMAS LN               | NORTH POLE      |
 VIC                    | 1111                | AUS              |
+-----------+-------------------+----------+---------------+-------------+-----------+
```

```
+----------+-------------+-------+-----------------------------+-------------+
|Confidence |StreetName     |HouseNumber |     AddressLine1
  | AddressType   |
+----------+-------------+-------+-----------------------------+-------------+
| 100.00   | MERIVALE      | 18    | 18 MERIVALE ST               | S
      |
| 99.42    | SERPENTINE    | 19       | 19 SERPENTINE RD E           |
 S       |
| 97.95    | VICTORIA      | 317      | 317 VICTORIA ST              |
 S       |
| 100.00   | O'CONNELL     | 16-18    | DUP 6 16-18 O'CONNELL ST     |
 S       |
| 0.00     | BRYGON CREEK  | 470      | LOT 154 470 BRYGON CREEK DR  |
 U       |
| 76.99    | GREENE        | 16        | 16 GREENE ST
| S       |
| 100.00   | BLAIRMOUNT    | 16       | U 47 16 BLAIRMOUNT ST        |
 S       |
```

```
| 100.00    | FRANCESCO      | 13-15     | 13-15 FRANCESCO CRES          |
 S        |
| 100.00    | RYANS          | 4         | 4 RYANS LANE                  |
 S        |
| 0.00      | CHRISTMAS      | 1         | 1 CHRISTMAS LN                |
 U        |
+-----------+----------------+-----------+-------------------------------+--------------+
```

Validate Address Global

サンプル Hive スクリプト

```
-- Register Universal Addressing Module [UAM-Global] BDQ Hive UDAF Jar

ADD JAR <Directory path>/uam.global.hive.${project.version}.jar;

ADD FILE <Directory path>/libAddressDoctor5.so;

-- Provide alias to UDAF class (optional). String in quotes represent
class names needed for this job to run.
CREATE TEMPORARY FUNCTION globalvalidation as
'com.pb.bdq.uam.process.hive.global.GlobalAddressingUDAF';

set hive.map.aggr = false;

-- set engine configuration
set pb.bdq.uam.global.engine.configurations=[{ "referenceData":
{"dataDir":"/media/New Volume/hduser/resources/uam/addressDoctor/5.8.0/",
 "referenceDataPathLocation":"LocaltoDataNodes"},
"databaseType":"BATCH_INTERACTIVE", "preloadingType":"NONE",
"allCountries":false, "supportedCountries":"CAN,USA,AUS"}];

-- set input configuration
set
pb.bdq.uam.global.input.configuration={"resultStateProvinceType":"COUNTRY_STANDARD",
 "processMatchingScope":"ALL", "processEnrichmentAMAS":false,
"inputForceCountryISO3":"AUS", "inputDefaultCountryISO3":"AUS",
"inputFormatDelimiter":"CRLF", "resultFormatDelimiter":"CRLF",
"resultIncludeInputs":false, "resultCountryType":"NAME_EN",
"processOptimizationLevel":"STANDARD",
"resultPreferredLanguage":"DATABASE", "processMode":"BATCH",
"resultPreferredScript":"DATABASE", "resultMaximumResults":1,
"resultCasing":"NATIVE",
"properties":{"Result.StateProvinceType":"COUNTRY_STANDARD",
"Process.MatchingScope":"ALL", "Process.EnrichmentAMAS":"false",
"Input.ForceCountryISO3":"AUS", "Input.FormatDelimiter":"CRLF",
"Result.FormatDelimiter":"CRLF", "Input.DefaultCountryISO3":"AUS",
```

```
"Result.IncludeInputs":"false", "Result.CountryType":"NAME_EN",
"Process.OptimizationLevel":"STANDARD",
"Result.PreferredLanguage":"DATABASE", "Process.Mode":"BATCH",
"Result.PreferredScript":"DATABASE", "Result.MaximumResults":"1",
"Result.Casing":"NATIVE", "Database.AddressGlobal":"Database"}};

-- set general configuration
set pb.bdq.uam.global.general.configuration={"cacheSize":"LARGE",
"maxThreadCount":8, "maxAddressObjectCount":8, "rangesToExpand":"NONE",
 "flexibleRangeExpansion":"ON", "enableTransactionLogging":false,
"maxMemoryUsageMB":1024};

-- set unlock codec
set pb.bdq.uam.global.unlockCode=<Insert your Unlock Code here>;

-- set header
set
pb.bdq.header=recordid,AddressLine1,City,StateProvince,PostalCode,Country;

-- Execute Query on the desired table, to display the job output on
console. This query returns a map of key value pairs containing output
 fields for each row.
SELECT tmp2.record["HouseNumber"], tmp2.record["Confidence"],
tmp2.record["AddressLine1"], tmp2.record["StreetName"],
tmp2.record["PostalCode"], tmp2.record["ElementInputStatus"],
tmp2.record["MailabilityScore"] FROM ( SELECT  globalvalidation(recordid,
 addressline1, city, stateprovince, postalcode, country) as mygp from
address) as addressgroup LATERAL VIEW explode(addressgroup.mygp) tmp2
as record ;


-- Query to dump output data to a file
INSERT OVERWRITE LOCAL DIRECTORY '/home/hadoop/GlobalAddressing/' row
format delimited FIELDS TERMINATED BY ',' lines terminated by '\n' STORED
 AS TEXTFILE
SELECT tmp2.record["HouseNumber"], tmp2.record["Confidence"],
tmp2.record["AddressLine1"], tmp2.record["StreetName"],
tmp2.record["PostalCode"], tmp2.record["ElementInputStatus"],
tmp2.record["MailabilityScore"] FROM ( SELECT  globalvalidation(recordid,
 addressline1, city, stateprovince, postalcode, country) as mygp from
address) as addressgroup LATERAL VIEW explode(addressgroup.mygp) tmp2
as record ;

+------------+--------------------+-----------+----------------+-------------+------------+
| address.recordid  |     address.addressline1     |  address.city   |
 address.stateprovince  | address.postalcode  | address.country  |
+------------+--------------------+-----------+----------------+-------------+------------+
| 1                 | 18 Merivale St               | South Brisbane  |
 QLD                    | 4101                | AUS              |
| 2                 | 19 Serpentine Rd             | Albany          |
 WA                     | 6330                | AUS              |
| 3                 | 317 VICTORIA ST GR           | BRUNSWICK       |
 VIC                    | 3056                | AUS              |
```

```
| 4                  | DUPLEX 6/16-18 O'CONNELL ST  | AINSLIE          |
 ACT                     | 2602                  | AUS                |
| 5                  | LOT 154 470 BRYGON CREEK DR  | UPPER COOMERA    |
 QLD                     | 4209                  | AUS                |
| 6                  | 16 GREENE ST                 | WARRAWONG        |
 ACT                     | 2502                  | AUS                |
| 7                  | UNIT 47/16 BLAIRMOUNT ST     | PARKINSON        |
 QLD                     | 4115                  | AUS                |
| 8                  | 13-15 FRANCESCO CRES         | BELLA VISTA      |
 NSW                     | 2153                  | AUS                |
| 9                  | 4 RYANS LANE                 | HEATHCOTE        |
 VIC                     | 3523                  | AUS                |
| 10                 | 1 CHRISTMAS LN               | NORTH POLE       |
 VIC                     | 1111                  | AUS                |
+--------------+---------------------+-----------+----------------+--------------+-------------+



+----------+--------------+-------+-------------------------------+--------------+
|Confidence |StreetName     |HouseNumber |     AddressLine1
  | AddressType   |
+----------+--------------+-----------+-------------------------+--------------+
| 100.00   | MERIVALE      | 18    | 18 MERIVALE ST               | S
      |
| 99.42    | SERPENTINE    | 19       | 19 SERPENTINE RD E          |
 S       |
| 97.95    | VICTORIA      | 317      | 317 VICTORIA ST             |
 S       |
| 100.00   | O'CONNELL     | 16-18    | DUP 6 16-18 O'CONNELL ST    |
 S       |
| 0.00     | BRYGON CREEK  | 470      | LOT 154 470 BRYGON CREEK DR  |
 U       |
| 76.99    | GREENE        | 16        | 16 GREENE ST
| S       |
| 100.00   | BLAIRMOUNT    | 16       | U 47 16 BLAIRMOUNT ST       |
 S       |
| 100.00   | FRANCESCO     | 13-15    | 13-15 FRANCESCO CRES        |
 S       |
| 100.00   | RYANS         | 4        | 4 RYANS LANE                |
 S       |
| 0.00     | CHRISTMAS     | 1        | 1 CHRISTMAS LN              |
 U       |
```

```
+----------+-------------+-----------+--------------------------+--------------+
```

Validate Address Loqate

サンプル Hive スクリプト

```
-- Register Universal Address Module [UAM] BDQ Hive Loqate UDAF Jar
ADD JAR <Directory path>/uam.loqate.hive.${project.version}.jar;

-- Provide alias to UDAF class (optional). String in quotes represent
class names needed for this job to run.
CREATE TEMPORARY FUNCTION loqatevalidation as
'com.pb.bdq.uam.process.hive.loqate.LoqateAddressingUDAF';

-- Adding required files to distributed cache.
ADD FILES <Directory Path>/loqate-core.car;
ADD FILES <Directory Path>/LoqateVerificationLevel.csv;
ADD FILES <Directory Path>/Loqate.csv;
ADD FILES <Directory Path>/countryTables.csv;
ADD FILES <Directory Path>/countryNameTables.csv;

set hive.map.aggr = false;

-- set process configuration
set pb.bdq.uam.loqate.process.configuration={"processType":"VALIDATE",
 "includeMatchedAddressElements":true,
"standardizedInputAddressElements":true, "returnAddressDataBlocks":true,
 "casing":"Mixed", "outputReportSummary":false,
"returnMultipleAddresses":false, "failedOnMultiMatchFound":false,
"countryFormat":"ENGLISH", "defaultCountry":"USA",
"scriptAlphabet":"Native", "returnGeocodedAddressFields":true,
"acceptanceLevel":"Level0", "minimumMatchScore":0,
"formatDataUsingAMASConventions":false,
"singleFieldDuplicateHandling":false,
"multiFieldDuplicateHandling":false,
"nonStandardFieldDuplicateHandling":false,
"outputFieldDuplicateHandling":false, "includeStandardAddress":true,
"duplicateHandling":false, "returnMultipleAddressCount":10};

-- set general configuration
set pb.bdq.uam.loqate.general.configuration={"maxIdle":null,
"minIdle":16, "maxActive":16, "maxWait":null, "whenExhaustedAction":null,
 "testOnBorrow":null, "testOnReturn":null, "testWhileIdle":null,
"timeBetweenEvictionRunsMillis":null, "numTestsPerEvictionRun":null,
"minEvictableIdleTimeMillis":null};

-- set engine configuration
```

```
set pb.bdq.uam.loqate.engine.configuration={"verbose":true,
"toolInfo":true, "outputAddressFormat":false, "logInput":false,
"logOutput":false, "logFileName":null, "matchScoreAbsoluteThreshold":60,
 "matchScoreThresholdFactor":95, "postalCodeMaxResults":10,
"strictReferenceMatch":false};

-- set reference directory path
set pb.bdq.referencedata.dir=/media/New
Volume/hduser/resources/uam/loqate/Linux;

-- set process type
set pb.bdq.uam.loqate.process.type=VALIDATE;

-- set input header
set pb.bdq.header='InputKeyValue,AddressLine1,AddressLine2,AddressLine3,
AddressLine4,City,StateProvince,PostalCode,Country,FirmName';

select SELECT tmp2.record["HouseNumber"], tmp2.record["Confidence"],
tmp2.record["AddressLine1"], tmp2.record["StreetName"],
tmp2.record["PostalCode"], tmp2.record["DPID"], tmp2.record["Barcode"]
 FROM ( SELECT  loqatevalidation(recordid, addressline1, city,
stateprovince, postalcode, country) as mygp from address) as <TABLE_NAME>
 LATERAL VIEW explode(addressgroup.mygp) tmp2 as record ;

-- Query to dump output data to a file
INSERT OVERWRITE LOCAL DIRECTORY '/home/hadoop/loqate/' row format
delimited FIELDS TERMINATED BY ',' lines terminated by '\n' STORED AS
TEXTFILE SELECT * FROM ( SELECT tmp2.record["HouseNumber"],
tmp2.record["Confidence"], tmp2.record["AddressLine1"],
tmp2.record["StreetName"], tmp2.record["PostalCode"],
tmp2.record["DPID"], tmp2.record["Barcode"] FROM ( SELECT
loqatevalidation(recordid, addressline1, city, stateprovince, postalcode,
 country) as mygp from address) as <TABLE_NAME> LATERAL VIEW
explode(addressgroup.mygp) tmp2 as record ;



--Sample Input
+-------------+-------------------------------+-------------+-----------+-------------+
| inputkeyvalue  |            addressline1            | stateprovince
 | postalcode  |    country    |
+-------------+-------------------------------+-------------+-----------+-------------+
| 1              | 80 Quan Su                          |
 |             | Vietnam       |
| 2              | Final Av. Panteón Foro Libertador  |
 | 1010        | Venezuela     |
| 3              | P O Box 834                         |
 |             | St Vincent    |
| 4              | Colonia 2066                        |
 |             | Uruguay       |
| 5              | Ave de la Resistance BP127          |
 |             | Burkina Faso  |
| 6              | Buyuk Turon Street, 41              |
```

```
 |             | Uzbekistan    |
| 7             | Empire State Building             | NY
 | 10118       | US            |
| 8             | 3 Leontovycha St                  |
 |             | Ukraine       |
| 9             |                                   | Ceredigion
 |             | Wales         |
| 10            | 5 Main Street                     | Ballindalloch
 |             | Scotland      |
+-------------+---------------------------+-----------+---------+-----------+


-- Sample Output
+-----------+--------------+--------+---+-----------------------------------+
|Match Score|StreetName     |HouseNumber |            addressline1
        |
+-----------+--------------+-----------+-----------------------------------+
| 100.00    | MERIVALE      | 80     | 80 Quan Su
   |
| 100.00     | SERPENTINE    |         | Final Av. Panteón Foro Libertador
   |
| 0.00     | VICTORIA      | 0        | P O Box 834
    |
| 75.00    | O'CONNELL     | 2066     | Colonia 2066
    |
| 83.33      | BRYGON CREEK   | 470       | Ave de la Resistance BP127
       |
| 100.00     | GREENE         |          | Buyuk Turon Street, 41
      |
| 96.8254    | BLAIRMOUNT     | 41        | Empire State Building
       |
| 83.950    | FRANCESCO      | 350      | 3 Leontovycha St
     |
| 50.00    | RYANS          | 3         |
     |
| 100      | CHRISTMAS      | 5         | 5 Main Street
     |
+-----------+--------------+-----------+-----------------------------------+

!quit
```

Universal Name モジュールの関数

Open Name Parser

サンプル Hive スクリプト

```
-- Register Universal Name Module [UNM] BDQ Hive UDF Jar
ADD JAR <Directory path>/unm.hive.${project.version}.jar;

-- Provide alias to UDF class (optional). String in quotes represent
class names needed for this job to run.
-- Open Name Parser is implemented as a UDF (User Defined function).
Hence it processes one row at a time and generates a map of key value
pairs for each row.
CREATE TEMPORARY FUNCTION opennameparser as
'com.pb.bdq.unm.process.hive.opennameparser.OpenNameParserUDF';

-- set rule
set rule='{"name":"name", "culture":"", "splitConjoinedNames":false,
"shortcutThreshold":0, "parseNaturalOrderPersonalNames":false,
"naturalOrderPersonalNamesPriority":1,
"parseReverseOrderPersonalNames":false,
"reverseOrderPersonalNamesPriority":2, "parseConjoinedNames":false,
"naturalOrderConjoinedPersonalNamesPriority":3,
"reverseOrderConjoinedPersonalNamesPriority":4,
"parseBusinessNames":false, "businessNamesPriority":5}';

-- Set Reference Directory. This must be a local path on cluster machines
 and must be present at the same path on each node of the cluster.
set refdir='/home/hadoop/reference/';

-- set header
set header='inputrecordid,Name,nametype';

-- Execute Query on the desired table, to display the job output on
console. This query returns a map of key value pairs containing output
 fields for each row.
select adTable.adid["Name"], adTable.adid["NameScore"],
adTable.adid["CultureCode"] from (select opennameparser(${hiveconf:rule},
 ${hiveconf:refdir}, ${hiveconf:header}, inputrecordid, name, nametype)
 as tmp1 from nameparser) as tmp LATERAL VIEW explode(tmp1) adTable AS
 adid;
```

```
-- Query to dump output data to a file
INSERT OVERWRITE LOCAL DIRECTORY '/home/hadoop/opennameparser/' row
format delimited FIELDS TERMINATED BY ',' lines terminated by '\n' STORED
 AS TEXTFILE
select adTable.adid["Name"], adTable.adid["NameScore"],
adTable.adid["CultureCode"] from (select opennameparser(${hiveconf:rule},
 ${hiveconf:refdir}, ${hiveconf:header}, inputrecordid, name, nametype)
 as tmp1 from nameparser) as tmp LATERAL VIEW explode(tmp1) adTable AS
 adid;



--sample input data
+-------------------------------+------------------------+----------------------+
| inputrecordid        | name           |      nametype          |
+-------------------------------+------------------------+----------------------+
| 1                                  |JOHN VAN DER LINDEN-JONES |
Simple Name    |
| 2                                  |RYAN JOHN SMITH           |  Simple
 Name      |
+-------------------------------+------------------------+----------------------+

--sample output data
+----------------------------+--------------+----------------------+
| Name           | NameScore    |      CultureCode    |
+----------------------------+--------------+----------------------+
| JOHN VAN DER LINDEN-JONES  |   75         |       True           |
| RYAN JOHN SMITH            |   100        |       True           |
+----------------------------+-----+--------------------------------+
```

付録

このセクションの構成

A - 例外

このセクションの構成

例外メッセージ

例外 - Java API

- `<Classname>.<Member>` is null or empty.
- `GroupbyMROption.numReduceTasks = 0` min values should be 1.
- `maxNumOfDuplicates` = 0 min values should be 1.
- No files available in the specified path.
- Unable to identify the input file as either Suspect or Candidate File.
- ExpressMatchKey defined but not available for the record\t
- Unable to get the FileName of the InputSplit.
- Unable to initialize engine.
- Error processing consolidated records:

例外 - Hive ユーザ定義関数

- _FUNC_ must have the minimum arguments.
- Unable to initialize engine.Rule passed: `<Rule used>`
- Expected argument type: String.Received argument type: `<Mismatched Type>`
- Exception: `<Header string>` configuration missing.
- Error processing consolidated records: `<Exception details>`
- Exception: Sort field column `<column name>` missing from job configuration.

B - 列挙体

このセクションの構成

一般的な列挙

列挙 MatchingAlgorithm

パッケージ: `com.pb.bdq.api.matcher`

クラス: `Algorithm`

1. Acronym (頭字語)
2. CharacterFrequency (文字出現回数)
3. DaitchMokotoffSoundex (Daitch-Mokotoff Soundex)
4. Date (日付)
5. DoubleMetaphone (Double Metaphone)
6. EditDistance (編集距離)
7. EuclideanDistance (ユークリッド距離)
8. ExactMatch (完全一致)
9. Initials (頭文字)
10. JaroWinklerDistance (Jaro-Winker 距離)
11. KeyboardDistance (キーボード距離)
12. Koeln
13. KullbackLeiblerDistance (Kullback-Liebler 距離)
14. Metaphone
15. SpanishMetaphone (スペイン語 Metaphone)
16. Metaphone3
17. NGramDistance (NGram 距離)
18. NGramSimilarity
19. NumericString (数値文字列)
20. Nysiis
21. Phonix
22. Soundex
23. SubString (部分文字列)
24. SyllableAlignment (シラブル配置)

列挙 Algorithm

パッケージ: `com.pb.bdq.api.matchkeygenerator`

クラス: `MatchKeyRule`

1. Soundex
2. Metaphone

3. SpanishMetaphone (スペイン語 Metaphone)
4. DoubleMetaphone (Double Metaphone)
5. Nysiis
6. Phonix
7. Metaphone3
8. Koeln
9. Consonant (子音)
10. SubString (部分文字列)

列挙 RecordSeparator

パッケージ: `com.pb.bdq.common.job`

クラス: `FilePath`

1. WINDOWS
2. LINUX
3. MACINTOSH

列挙 ReferenceDataPathLocation

パッケージ: `com.pb.bdq.common.job`

Enum 定数	説明
HDFS	リファレンス データは HDFS ディレクトリに配置されます。
LocaltoDataNodes	リファレンス データはクラスタ内の使用可能なすべてのデータ ノードに配置されます。

列挙 Operation

パッケージ: `com.pb.bdq.api.consolidation`

1. CONTAINS
2. HIGHEST
3. LOWEST
4. NOT_EQUAL
5. GREATER
6. LESSER
7. EQUAL
8. GREATER_THAN_EQUAL_TO
9. LESS_THAN_EQUAL_TO
10. IS_EMPTY
11. IS_NOT_EMPTY
12. MOST_COMMON

13. LONGEST
14. SHORTEST

列挙 MatchingMethod

パッケージ: `com.pb.bdq.api.matcher`

クラス: `ParentMatchRule`

1. AllTrue
2. AnyTrue
3. BasedOnThreshold

列挙 ScoringMethod

パッケージ: `com.pb.bdq.api.matcher`

クラス: `MatchRule`

1. Minimum
2. Maximum
3. Average
4. WeightedAverage
5. VectorSummation

列挙 MissingDataMethod

パッケージ: `com.pb.bdq.api.matcher`

クラス: `MatchRule`

1. IgnoreBlanks
2. CountAs100
3. CountAs0
4. CompareBlanks

列挙 JoinType

パッケージ: `com.pb.bdq.api.consolidation`

クラス: `ConjoinedRule`

1. OR
2. AND

列挙 IncludeTerm

パッケージ: `com.pb.bdq.api.advtransformer`

クラス: `TableDataExtraction`

1. ExtractedData
2. NonExtractedData

3. TermNeither

列挙 Extract
パッケージ: com.pb.bdq.api.advtransformer

クラス: TableDataExtraction

1. ExtractTerm
2. ExtractNWordsLeft
3. ExtractNWordsRight

列挙 AdvTransformerExtractionType
パッケージ: com.pb.bdq.api.advtransformer

クラス: AbstractAdvancedTransformerRules

1. TableData
2. RegularExpression

列挙 MatchRuleType
パッケージ: com.pb.bdq.api.matcher

クラス: MatchRule

1. Parent
2. Child

列挙 SortInput
パッケージ: com.pb.bdq.api.matcher

クラス: MatchRule

1. CHARS
2. TERMS

列挙 TableLookupAction
パッケージ: com.pb.bdq.api.tablelookup

クラス: AbstractTableLookupRule

1. Standardize
2. Categorize
3. Identify

Universal Addressing 列挙体

列挙 DatabaseType
パッケージ: com.pb.bdq.api.uam.global

クラス: GlobalAddressingEngineConfiguration

1. BATCH_INTERACTIVE
2. FASTCOMPLETION
3. CERTIFIED

列挙 PreloadingType
パッケージ: com.pb.bdq.api.uam.global

クラス: GlobalAddressingEngineConfiguration

1. NONE
2. FULL
3. PARTIAL

列挙 CountryCodes
パッケージ: com.pb.bdq.api.uam

説明: サポートされているすべての国に割り当てられる英文字のコード。

列挙 StateProvinceType
パッケージ: com.pb.bdq.api.uam.global

インターフェイス: GlobalAddressingInputOption

1. COUNTRY_STANDARD
2. ABBREVIATION
3. EXTENDED

列挙 CountryType
パッケージ: com.pb.bdq.api.uam.global

インターフェイス: GlobalAddressingInputOption

1. ISO2
2. ISO3
3. ISO_NUMBER
4. NAME_CN
5. NAME_DA
6. NAME_DE

7. NAME_EN
8. NAME_ES
9. NAME_FI
10. NAME_FR
11. NAME_GR
12. NAME_HU
13. NAME_IT
14. NAME_JP
15. NAME_KR
16. NAME_NL
17. NAME_PL
18. NAME_PT
19. NAME_RU
20. NAME_SA
21. NAME_SE

列挙 PreferredScript

パッケージ: `com.pb.bdq.api.uam.global`

インターフェイス: `GlobalAddressingInputOption`

1. DATABASE
2. POSTAL_ADMIN_PREF
3. POSTAL_ADMIN_ALT
4. LATIN
5. LATIN_ALT
6. ASCII_SIMPLIFIED
7. ASCII_EXTENDED

列挙 PreferredLanguage

パッケージ: `com.pb.bdq.api.uam.global`

インターフェイス: `GlobalAddressingInputOption`

1. DATABASE
2. ENGLISH

列挙 Casing

パッケージ: `com.pb.bdq.api.uam.global`

インターフェイス: `GlobalAddressingInputOption`

1. NATIVE
2. UPPER
3. LOWER

4. MIXED
5. NOCHANGE

列挙 OptimizationLevel

パッケージ: com.pb.bdq.api.uam.global

インターフェイス: GlobalAddressingInputOption

1. NARROW
2. STANDARD
3. WIDE

列挙 Mode

パッケージ: com.pb.bdq.api.uam.global

インターフェイス: GlobalAddressingInputOption

1. BATCH
2. CERTIFIED
3. FASTCOMPLETION
4. INTERACTIVE
5. PARSE

列挙 MatchingScope

パッケージ: com.pb.bdq.api.uam.global

インターフェイス: GlobalAddressingInputOption

1. ALL
2. LOCALITY_LEVEL
3. STREET_LEVEL
4. DELIVERYPOINT_LEVEL

列挙 FormatDelimiter

パッケージ: com.pb.bdq.api.uam.global

インターフェイス: GlobalAddressingInputOption

1. CRLF
2. LF
3. CR
4. SEMICOLON
5. COMMA
6. TAB
7. PIPE
8. SPACE

列挙 ExhaustedAction
パッケージ: `com.pb.bdq.api.uam.loqate`

クラス: `LoqateAddressingGeneralConfiguration`

1. GROW
2. BLOCK
3. FAIL

列挙 AcceptanceLevel
パッケージ: `com.pb.bdq.api.uam.loqate.validate`

クラス: `LoqateAddressingValidateConfiguration`

1. Level0
2. Level1
3. Level2
4. Level3
5. Level4
6. Level5

列挙 OutputCasing
パッケージ: `com.pb.bdq.api.uam.loqate.validate`

クラス: `LoqateAddressingValidateConfiguration`

1. Mixed
2. Upper

列挙 CountryFormat
パッケージ: `com.pb.bdq.api.uam.loqate.validate`

クラス: `LoqateAddressingValidateConfiguration`

1. ENGLISH
2. ISO
3. UPU

列挙 ScriptAlphabet
パッケージ: `com.pb.bdq.api.uam.loqate.validate`

クラス: `LoqateAddressingValidateConfiguration`

1. InputScript
2. Native
3. Latin_English

列挙 CacheSize
パッケージ: com.pb.bdq.api.uam.global

クラス: GlobalAddressingGeneralConfiguration

1. NONE
2. SMALL
3. LARGE

列挙 RangesToExpand
パッケージ: com.pb.bdq.api.uam.global

クラス: GlobalAddressingGeneralConfiguration

1. NONE
2. ONLY_WITH_VALID_ITEMS

列挙 FlexibleRangeExpansion
パッケージ: com.pb.bdq.api.uam.global

クラス: GlobalAddressingGeneralConfiguration

1. ON
2. OFF

列挙 CasingType
パッケージ: com.pb.bdq.api.universaladdress

クラス: UniversalAddressInputConfiguration

1. MIXED
2. UPPER

列挙 CityNameFormat
パッケージ: com.pb.bdq.api.universaladdress

クラス: UniversalAddressInputConfiguration

1. CITY_FORMAT_LONG
2. CITY_FORMAT_SHORT
3. CITY_FORMAT_STANDARD

列挙 OutputCountryFormat
パッケージ: com.pb.bdq.api.universaladdress

クラス: UniversalAddressInputConfiguration

1. ENGLISH
2. FRENCH

3. GERMAN
4. SPANISH
5. ISO
6. UPU

列挙 DualAddressLogic
パッケージ: `com.pb.bdq.api.universaladdress`

クラス: `UniversalAddressInputConfiguration`

1. DUAL_NORMAL
2. DUAL_PO_BOX
3. DUAL_STREET

列挙 StandardAddressFormat
パッケージ: `com.pb.bdq.api.universaladdress`

クラス: `UniversalAddressInputConfiguration`

1. STANDARD_ADDRESS_FORMAT_COMBINED_UNIT
2. STANDARD_ADDRESS_FORMAT_SEPARATE_UNIT
3. STANDARD_ADDRESS_FORMAT_SEPARATE_DUAL

列挙 StreetMatchingStrictness
パッケージ: `com.pb.bdq.api.universaladdress`

クラス: `UniversalAddressInputConfiguration`

1. MATCHING_STRICTNESS_EQUAL
2. MATCHING_STRICTNESS_TIGHT
3. MATCHING_STRICTNESS_MEDIUM
4. MATCHING_STRICTNESS_LOOSE

列挙 FirmMatchingStrictness
パッケージ: `com.pb.bdq.api.universaladdress`

クラス: `UniversalAddressInputConfiguration`

1. MATCHING_STRICTNESS_EQUAL
2. MATCHING_STRICTNESS_TIGHT
3. MATCHING_STRICTNESS_MEDIUM
4. MATCHING_STRICTNESS_LOOSE

列挙 DirectionalMatchingStrictness
パッケージ: `com.pb.bdq.api.universaladdress`

クラス: `UniversalAddressInputConfiguration`

1. MATCHING_STRICTNESS_EQUAL
2. MATCHING_STRICTNESS_TIGHT
3. MATCHING_STRICTNESS_MEDIUM
4. MATCHING_STRICTNESS_LOOSE

列挙 StandardAddressPMBLine

パッケージ: `com.pb.bdq.api.universaladdress`

クラス: `UniversalAddressInputConfiguration`

1. STANDARD_ADDRESS_PMB_LINE_NONE
2. STANDARD_ADDRESS_PMB_LINE_1
3. STANDARD_ADDRESS_PMB_LINE_2

列挙 PreferredCity

パッケージ: `com.pb.bdq.api.universaladdress`

クラス: `UniversalAddressInputConfiguration`

1. CITY_OVERRIDE_NAME_ZIP4
2. CITY_USPS_STATE_FILE
3. CITY_PRIMARY_NAME

列挙 DPVFileType

パッケージ: `com.pb.bdq.api.universaladdress`

クラス: `UniversalAddressGeneralConfiguration`

1. SPLIT
2. FULL
3. FLAT

列挙 DPVMemoryModel

パッケージ: `com.pb.bdq.api.universaladdress`

クラス: `UniversalAddressGeneralConfiguration`

1. PICO
2. MICRO
3. SMALL
4. MEDIUM
5. LARGE
6. HUGE

列挙 LacsLinkMemoryModel

パッケージ: `com.pb.bdq.api.universaladdress`

クラス: `UniversalAddressGeneralConfiguration`

1. PICO
2. MICRO
3. SMALL
4. MEDIUM
5. LARGE
6. HUGE

列挙 SuiteLinkMemoryModel

パッケージ: `com.pb.bdq.api.universaladdress`

クラス: `UniversalAddressGeneralConfiguration`

1. PICO
2. MICRO
3. SMALL
4. MEDIUM
5. LARGE
6. HUGE

列挙 DPVSuccessStatusCondition

パッケージ: `com.pb.bdq.api.universaladdress`

クラス: `UniversalAddressInputConfiguration`

1. DPV_CONDITON_FULL
2. DPV_CONDITON_PARTIAL
3. DPV_CONDITON_ALWAYS

列挙 UAMCASSReportType

パッケージ: `com.pb.bdq.uam.common`

1. CASS_3553
2. CASS_DETAIL
3. CASS_DETAIL2
4. CASS_DETAIL3

C - ISO 国コードとモジュール サポート

このセクションの構成

著作権に関する通知

USPS® 情報

Pitney Bowes Inc. は、ZIP + 4® データベースを光学および磁気媒体に発行および販売する非独占的ライセンスを所有しています。CASS、CASS 認定、DPV、eLOT、FASTforward、First-Class Mail、Intelligent Mail、LACS^{Link}、NCOA^{Link}、PAVE、PLANET Code、Postal Service、POSTNET、Post Office、RDI、Suite^{Link}、United States Postal Service、Standard Mail、United States Post Office、USPS、ZIP Code、および ZIP + 4 の各商標は United States Postal Service が所有します。United States Postal Service に帰属する商標はこれに限りません。

Pitney Bowes Inc. は、NCOA^{Link}® 処理に対する USPS® の非独占的ライセンスを所有しています。

Pitney Bowes Software の製品、オプション、およびサービスの価格は、USPS® または米国政府によって規定、制御、または承認されるものではありません。RDI™ データを利用して郵便送料を判定する場合に、使用する郵便配送業者の選定に関するビジネス上の意思決定が USPS® または米国政府によって行われることはありません。

データ プロバイダおよび関連情報

このメディアに含まれて、Pitney Bowes Software アプリケーション内で使用されるデータ製品は、各種商標によって、および次の 1 つ以上の著作権によって保護されています。

© Copyright United States Postal Service.All rights reserved.

© 2014 TomTom.All rights reserved.TomTom および TomTom ロゴは TomTom N.V の登録商標です。

© 2016 HERE

Fuente: INEGI (Instituto Nacional de Estadística y Geografía)

電子データに基づいています。© National Land Survey Sweden.

© Copyright United States Census Bureau

© Copyright Nova Marketing Group, Inc.

このプログラムの一部は著作権で保護されています。© Copyright 1993-2007 by Nova Marketing Group Inc. All Rights Reserved

© Copyright Second Decimal, LLC

© Copyright Canada Post Corporation

この CD-ROM には、Canada Post Corporation が著作権を所有している編集物からのデータが収録されています。

© 2007 Claritas, Inc.

Geocode Address World データ セットには、**http://creativecommons.org/licenses/by/3.0/legalcode** に存在するクリエイティブ コモンズ アトリビューション ライセンス (「アトリビューション ライセンス」) の下に提供されている GeoNames Project (**www.geonames.org**) からライセンス供与されたデータが含まれています。お客様による GeoNames データ (Spectrum™ Technology Platformユーザ マニュアルに記載) の使用は、アトリビューション ライセンスの条件に従う必要があり、お客様と Pitney Bowes Software, Inc. との契約と、アトリビューション ライセンスの間に矛盾が生じる場合は、アトリビューション ライセンスのみに基づいてそれを解決する必要があります。お客様による GeoNames データの使用に関しては、アトリビューション ライセンスが適用されるためです。

3001 Summer Street
Stamford CT 06926-0700
USA

www.pitneybowes.com