Oracle® GoldenGate

フラット・ファイル・アダプタ管理者ガイド

11*g* リリース 2 (11.2.1.0.0)

B71935-01 (原本部品番号 : E28381-01)

2012 年 12 月

ORACLE

Oracle GoldenGate フラット・ファイル・アダプタ管理者ガイド 11g リリース 2 (11.2.1.0.0)

B71935-01 (原本部品番号 : E28381-01)

目次

第 1 章
概要

このガイドでは、フラット・ファイル用 GoldenGate のインストール、構成および実行について説明します。

Oracle GoldenGate

Oracle GoldenGate の基本的な機能は次のとおりです。

- トランザクションの変更をソース・データベースから取得します。大半のデータベースで、これはデータベース・トランザクション・ログの読取りによって行われます。
- Oracle GoldenGate 証跡と呼ばれる、データベースに依存しないファイルのセットとしてこれらの変更を送信およびキューイングします。
- オプションで、マッピング・パラメータおよび関数を使用してソース・データを変更します。
- 証跡のトランザクションをターゲット・システム・データベースに適用します。

Oracle GoldenGate は、異種データベースおよびオペレーティング・システム間でこの取得と適用をほぼリアルタイムで行います。

アダプタ統合オプション

Oracle GoldenGate アダプタは Oracle GoldenGate コア製品と統合されて、1) JMS メッセージを読み取り、Oracle GoldenGate 証跡として配信、2) Oracle GoldenGate 証跡を読み取り、トランザクションを JMS プロバイダ、他のメッセージング・システムまたはカスタム・アプリケーションに配信、または、3) Oracle GoldenGate 証跡を読み取り、他のアプリケーションで使用可能なフラット・ファイルにトランザクションを書込みのいずれかを行います。

トランザクションの証跡への取得

Oracle GoldenGate メッセージ取得アダプタを使用してキューからメッセージを読み取り、Oracle GoldenGate Extract プロセスと通信して、処理したデータを含む証跡を生成します。

メッセージ取得アダプタは、通常の Extract プロセスのベンダー・アクセス・モジュール (VAM) プラグインとして実装されます。プロパティ、ルールおよび外部ファイルのセットによってメッセージ接続情報が指定され、メッセージの解析方法とターゲット GoldenGate 証跡のレコードへのマップ方法が定義されます。

現在このアダプタでは、JMS テキスト・メッセージからの取得がサポートされています。

トランザクションの証跡からの適用

Oracle GoldenGate 配信アダプタを使用して、トランザクションの変更をリレーショナル・データベース以外のターゲット (DataStage、Ab Initio、Informatica などの ETL ツール、JMS メッセージング、カスタム API など) に適用します。Oracle GoldenGate との統合には、様々なオプションがあります。

- *フラット・ファイル統合*: 主にプロプライエタリ ETL やレガシー・アプリケーション用。フラット・ファイル用 Oracle GoldenGate で、バッチ・ファイルを入力とするツールによって消費されるマイクロ・バッチをディスクに書き込むことができます。データは、デリミタ区切り、長さ区切り、バイナリなどターゲット・アプリケーションの仕様に合せてフォーマットされます。バッチ・ファイル・ロールオーバーの時間ウィンドウを分または秒に短縮することで、これらのシステムへほぼリアルタイムでフィードされます。
- *メッセージング*: トランザクションまたは操作はメッセージ(XML形式など)としてJMSに発行されます。ActiveMQ、JBoss Messaging、TIBCO、WebLogic JMS、WebSphere MQ などの JMS プロバイダを構成できます。
- *Java API*: カスタム・イベント・ハンドラを Java で記述し、Oracle GoldenGate によってソース・システムで取得されたトランザクション、操作およびメタデータの変更を処理できます。これらのカスタム Java ハンドラで、ターゲット・システムで公開されているサードパーティ Java API にこれらの変更を適用します。

3 つのオプションとも、Oracle GoldenGate のユーザー・イグジット・インタフェースである C API を使用したコア Oracle GoldenGate 製品の拡張として実装されています。

- フラット・ファイル統合の場合、フラット・ファイル用 Oracle GoldenGate によって、Oracle GoldenGate Extract プロセスに動的にリンクされるユーザー・イグジット・ライブラリが提供されます。構成はプロパティ・ファイルを使用して行われ、プログラミングは必要ありません。
- JMS または Java API を使用した Java 統合の場合、Java 用 Oracle GoldenGate を使用します。

このガイドでは、フラット・ファイル用 Oracle GoldenGate の使用方法について説明します。

フラット・ファイル用 Oracle GoldenGate

フラット・ファイル用 Oracle GoldenGate は、Oracle GoldenGate によって取得されたトランザクション・データを、サードパーティ製品によって消費されるローリング・フラット・ファイルに出力します。フラット・ファイル用 Oracle GoldenGate は、Oracle GoldenGate Extract プロセスに統合される共有ライブラリ (.so または .dll) で提供されるユーザー・イグジットとして実装されます。

ユーザー・イグジットでは、2 つの出力モードがサポートされます。

- DSV: デリミタ (カンマなど) 区切りの値
- LDV: 長さ区切りの値

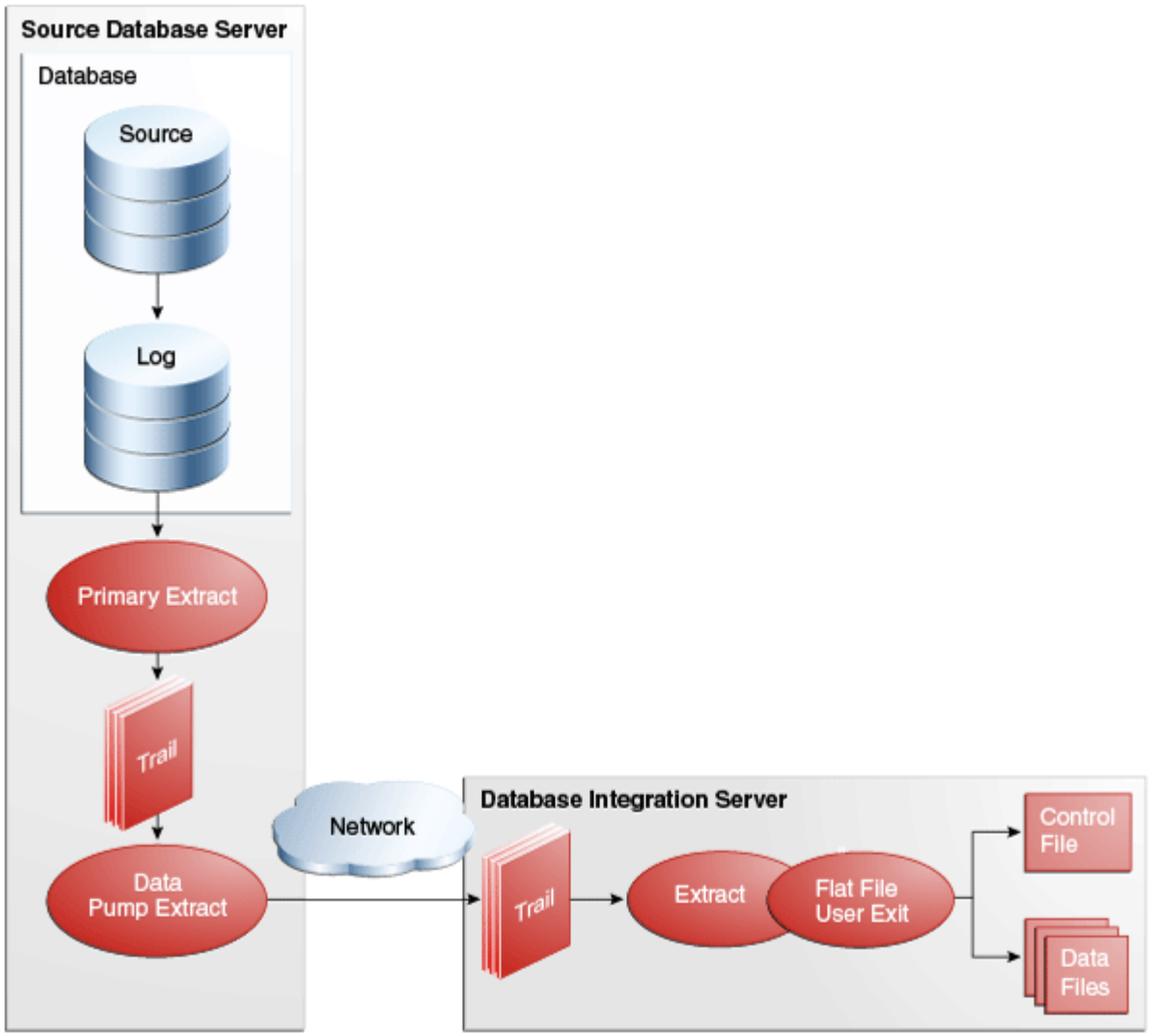
次のようにデータを出力できます。

- すべてを 1 ファイルに
- 表ごとに 1 ファイル
- 操作コードごとに 1 ファイル

ユーザー・イグジットは、時間またはサイズの基準に基づいてロールオーバーできます。ファイルをフラッシュし、Oracle GoldenGate のチェックポイントのたびにチェックポイントを保持してリカバリを保証します。サポートされているデータ統合製品との同期用にロールオーバーしたファイルのリスト

を含む制御ファイルを作成します。監査で使用するサマリー・ファイルを生成することもできます。その他のプロパティによって、フォーマット (デリミタやその他の値)、ディレクトリ、ファイル拡張子、メタデータ列 (表名、ファイルの場所など) およびデータ・オプションを制御します。

図 1　フラット・ファイル用 Oracle GoldenGate



Oracle GoldenGate のドキュメント

Oracle GoldenGate フラット・ファイル・アダプタまたは Java アダプタと組み合せて使用するコア Oracle GoldenGate 製品のインストールおよび構成については、Oracle GoldenGate ドキュメントを参照してください。

- *インストレーションおよびセットアップ・ガイド*: Oracle GoldenGate でサポートされているデータベースごとにこのガイドがあります。システム要件、インストール前とインストール後の処理、インストール手順や Oracle GoldenGate レプリケーション・ソリューションをインストールするためのシステム固有のその他の情報が含まれています。

- 『*Oracle GoldenGate Windows and UNIX 管理者ガイド*』: WindowsおよびUNIXプラットフォームで Oracle GoldenGate レプリケーション・ソリューションを計画、構成および実装する方法について説明します。
- 『*Oracle GoldenGate Windows and UNIX リファレンス・ガイド*』: Windows および UNIX プラットフォーム用の Oracle GoldenGate のパラメータ、コマンドおよび関数の詳細が含まれています。
- 『*Oracle GoldenGate Windows and UNIX トラブルシューティングおよびチューニング・ガイド*』: Oracle GoldenGate レプリケーション・ソリューションのパフォーマンス向上に関する推奨と、一般的な問題に対するソリューションが含まれています。

第 2 章

フラット・ファイル用 Oracle GoldenGate のインストール

この章では、Oracle GoldenGate フラット・ファイル・アダプタの新規インスタンスのインストール方法と既存インスタンスのアップグレード方法について説明します。

Oracle GoldenGate フラット・ファイル・アダプタのインストール

Oracle GoldenGate アダプタには、Windows 用、Linux 用および UNIX (32 ビットおよび 64 ビット) 用があります。http://edelivery.oracle.com で、使用しているオペレーティング・システムとアーキテクチャ用の Oracle GoldenGate フラット・ファイル・アダプタのビルドがあるかどうか確認してください。

インストールの概要

Oracle GoldenGate アダプタのインストール zip ファイルには、次のものが含まれています。

- Oracle GoldenGate Java アダプタ。このアダプタの詳細は、『*Java 用 Oracle GoldenGate* フラット・ファイル用管理者ガイド』を参照してください。
- Oracle GoldenGate フラット・ファイル・アダプタ。
- アダプタ実行版 Oracle GoldenGate。この版はデータベース固有ではないため、*汎用*と呼ばれることがありますが、プラットフォームには依存します。

JMS 取得用 Java アダプタは Oracle GoldenGate の汎用ビルドで実行される必要があります。ただし、ターゲットへの証跡データの配信用アダプタを使用する場合、汎用ビルドは必要ありません。この場合、Oracle GoldenGate フラット・ファイル・アダプタまたは Java アダプタを任意のデータベース版の Oracle GoldenGate とともに使用できます。Oracle GoldenGate の非汎用インスタンスをインストールする場合、まず、一時的な場所に解凍し、Oracle GoldenGate のインストール場所にアダプタ・ファイルをコピーします。

インストール手順

次の手順を実行して Oracle GoldenGate アダプタをインストールします。

1. 名前に空白を含まないインストール・ディレクトリを作成します。zip ファイルをこの新規インストール・ディレクトリに抽出します。次に例を示します。

```
Shell> mkdir {installation_directory}
Shell> cp path/to/{installation_zip} {installation_directory}
Shell> cd {installation_directory}
Shell> unzip {installation_zip}
```

Linux または UNIX の場合、次も必要です。

```
Shell> tar -xf {installation_tar}
```

2. インストール・ディレクトリから移動せずに GGSCI を起動し、インストール場所に残りのサブディレクトリを作成します。

```
Shell> ggsci
GGSCI> CREATE SUBDIRS
```

ディレクトリ構造

次の表は、インストール・ファイルの解凍とサブディレクトリの作成の結果、生成されるサブディレクトリとファイルを含む例です。次の表記規則が使用されています。

- サブディレクトリは、大カッコ ([]) で囲まれています。
- レベルは、パイプおよびハイフン (|、-) で示されています。
- **内部**という表記は、変更できない読取り専用のディレクトリを示します。
- テキスト・ファイル (*.txt) はリストに含まれていません。
- Oracle GoldenGate ユーティリティ (Defgen、Logdump、Keygen など) はリストに含まれていません。

表 1　　インストール・ディレクトリ構造の例

ディレクトリ	説明
[*gg_install_dir*]	Oracle GoldenGate インストール・ディレクトリ (Windows の場合の C:/ggs、UNIX の場合の /home/user/ggs など)。
|-ggsci	プロセスの起動、停止および管理に使用されるコマンドライン・インタフェース。
|-mgr	Manager プロセス。
|-extract	Java アプリケーションまたはフラット・ファイル・ライターを起動する Extract プロセス。
|-[AdapterExamples]	Java ユーザー・イグジット、Java API、フラット・ファイル・ライターおよび JMS 取得アダプタのサンプル構成ファイル。
|-[UserExitExamples]	サンプル C プログラミング言語ユーザー・イグジット・コード。
|-[dirprm]	ユーザーによって作成される、次のようなパラメータおよびプロパティ・ファイルをすべて含むサブディレクトリ。 ◆ javaue.prm ◆ javaue.proprties ◆ jmsvam.prm ◆ jmsvam.properties ◆ ffwriter.prm

表 1　　インストール・ディレクトリ構造の例（続き）

ディレクトリ	説明
`|-[dirdef]`	証跡のメタデータを定義するソース定義ファイル (*.def) を含むサブディレクトリ。 ◆ ユーザー・イグジット証跡データ用に Defgen コア・ユーティリティによって作成されます。 ◆ VAM メッセージ取得用に Gendef アダプタ･ユーティリティによって作成されます。
`|-[dirdat]`	VAM Extract によって生成され、ユーザー・イグジット Extract によって読み取られる証跡ファイルを含むサブディレクトリ。
`|-[dirchk]`	**内部**:チェックポイント・ファイルを含むサブディレクトリ。
`|-[dirpcs]`	**内部**:プロセス・ステータス・ファイルを含むサブディレクトリ。
`|-[dirjar]`	**内部**:Oracle GoldenGate Monitor jar ファイルを含むサブディレクトリ。
`|-[ggjava]`	**内部**:Java jar のインストール・ディレクトリ。
`|-ggjava.jar`	クラスパスおよび依存関係を定義するメイン Java アプリケーション jar。
`|-[resources]`	すべての `ggjava.jar` 依存ファイルを含むサブディレクトリ。次のサブディレクトリを含みます。 ◆ `[class]`: プロパティおよびリソース ◆ `[lib]`: `ggjava.jar` で必要とされるアプリケーション jar
`|-ggjava_ue.dll`	ユーザー・イグジット共有ライブラリ。これは、UNIX では `libggjava_ue.so` です。
`|-ggjava_vam.dll`	VAM 共有ライブラリ。これは、UNIX では `libggjava_vam.so` です。
`|-gendef`	JMS メッセージ入力のメタデータを含むアダプタ・ソース定義ファイルを生成するユーティリティ。これは、証跡の入力メタデータを含むソース定義を作成する Oracle GoldenGate Defgen ユーティリティとは異なることに注意してください。
`|-flatfilewriter.dll`	Oracle GoldenGate フラット・ファイル・アダプタ用の Windows の `.dll` ライブラリまたは UNIX の `.so` ライブラリ。
`|-. . .`	インストールに含まれていたり、後で作成されるその他のサブディレクトリおよびファイル。

Oracle GoldenGate フラット・ファイル・アダプタのアップグレード

次の手順を実行して、既存の Oracle GoldenGate フラット・ファイル・アダプタをアップグレードします。

1. 名前に空白を含まないインストール・ディレクトリを作成します。
2. zip ファイルをこの新規インストール・ディレクトリに抽出します。これによって、ファイルがいくつかのサブディレクトリにダウンロードされます。

3. インストール・ディレクトリから移動せずに GGSCI を起動し、インストール場所に残りのサブディレクトリを作成します。

```
Shell> ggsci
GGSCI> CREATE SUBDIRS
```

4. すべての dirprm ファイルを既存のインストールから新規インストール場所の dirprm ディレクトリにコピーします。

> **注意** **すべての構成ファイルは dirprm ディレクトリにある必要があります。プロパティ・ファイルまたは他の構成ファイルが、古いインストールの dirprm 以外の場所にある場合、新規インストールの dirprm ディレクトリにコピーします。**

5. すべての dirdef ファイルを既存のインストールから新規インストール場所の dirdef ディレクトリにコピーします。
6. 古いインストールに他のカスタム・ファイルがある場合、新規インストール・ディレクトリにコピーします。
7. ソース・データベース取得で新しい 11.2.1 形式で証跡に書き込む場合、ソース・データベースで Defgen を実行して、新しい形式のソース定義ファイルを作成します。その後、それらを dirdef にインストールします。

   ソース・データベース取得で以前の形式で証跡に書き込む場合、既存のソース定義ファイルを次のいずれにも使用できます。

   - 11.2.1 より前の Oracle GoldenGate アダプタでサポートされている証跡の形式であるリリース 9.5 形式。
   - リリース 11.2.1 の Oracle GoldenGate アダプタでサポートされている 10.1 以上の証跡の形式。

8. 新規インストール・ディレクトリの Extract ポンプ・プロセスを構成します。これは、GGSCI を起動して Extract を追加し、証跡を指定することで行います。

```
GGSCI> ADD EXTRACT group_name, EXTTRAILSOURCE trail_name, . . .
```

9. Extract プロセスを起動し、Extract プロセスが稼働していることを確認します。

```
GGSCI> START EXTRACT group_name
GGSCI> INFO EXTRACT group_name
GGSCI> VIEW REPORT group_name
```

10. 新規 Oracle GoldenGate アダプタ・インストール・ディレクトリに書き込むようにソース・システムを変更します。

    - (オプション) 使用しているデータベース・プラットフォーム用のアップグレード手順の後に、ソース・データベース Oracle GoldenGate 取得をアップグレードします。
    - 新規Oracle GoldenGateアダプタのインストール場所のdirdatディレクトリに書き込むようにソース・データベース取得を構成します。
    - 古い Oracle GoldenGate アダプタ・インストールですべてのデータの処理を終えたら、新しい場所にデータを送信するプロセスに切り替えます。

第 3 章

フラット・ファイル用 Oracle GoldenGate の構成と使用

この章では、ユーザー・イグジット・パラメータとファイル・ライターのプロパティを設定する、システムの構成について説明します。ファイルのロールオーバーの制御、統計の収集および Oracle GoldenGate アダプタ・インスタンスのトラブルシューティングによってシステムを管理する方法について説明します。

Oracle GoldenGate フラット・ファイル・アダプタの構成

12 ページの図「標準的な構成」に標準的なフラット・ファイル用 Oracle GoldenGate の構成を示します。

ソース・データベース・システムを構成するには、次のようにします。

```
GGSCI > ADD EXTRACT pump, EXTTRAILSOURCE dirdat/aa
GGSCI > ADD RMTTRAIL dirdat/bb, EXTRACT pump, MEGABYTES 20
```

データ統合を構成するには、次のようにします。

```
GGSCI > ADD EXTRACT ffwriter, EXTTRAILSOURCE dirdat/bb
```

前述のサンプル・プロセス名と証跡名は、任意の有効な名前で置き換えます。プロセス名は 8 文字以下、証跡名は 2 文字である必要があります。

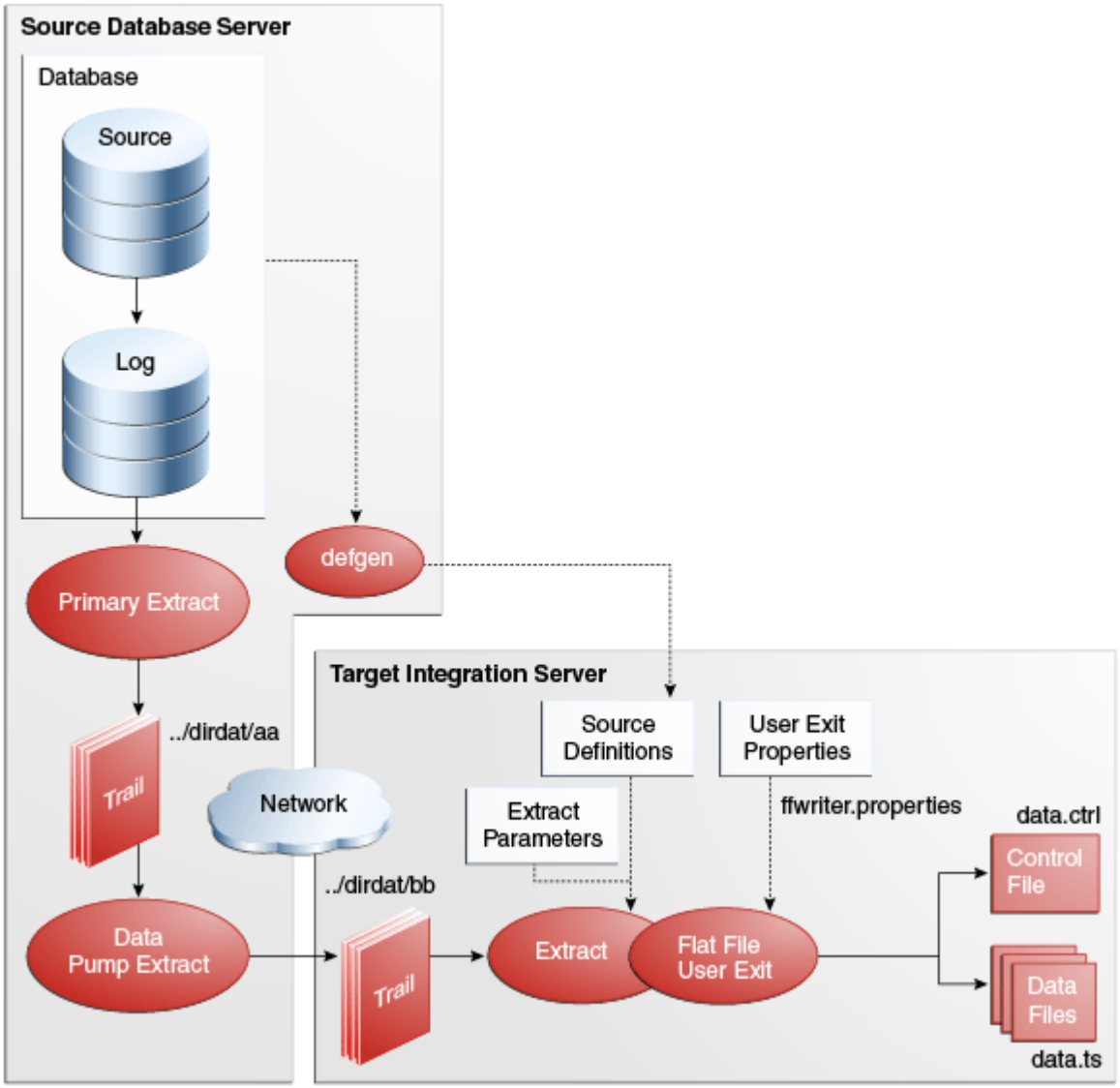
ユーザー・イグジット Extract のパラメータ

ユーザー・イグジット Extract のパラメータ (`ffwriter.prm`) は次のとおりです。

パラメータ	説明
`EXTRACT FFWRITER`	すべての Extract パラメータ・ファイルは Extract 名で始まります。この場合は、ユーザー・イグジットのファイル・ライター名です。
`SOURCEDEFS dirdef/hr_ora.def`	証跡の内容を決めるソース定義ファイル。
`CUSEREXIT flatfilewriter.dll CUSEREXIT PASSTHRU, INCLUDEUPDATEBEFORES, PARAMS ffwriter.properties`	`CUSEREXIT` パラメータのオプションは次のとおりです。 ◆ `flatfilewriter.dll` は、ユーザー・イグジット .dll または .so ライブラリの名前です。 ◆ `CUSEREXIT` は、起動されるユーザー・イグジット・ルーチンの名前です (大文字 / 小文字は区別されます)。

パラメータ	説明
	◆ PASSTHRU は、Extract プロセスが証跡に書き込む必要がないことを指定します。 ◆ INCLUDEUPDATEBEFORES は、ビフォア・イメージとアフター・イメージの両方を出力に含めることを許可します。整合性目的およびトランザクションの追跡用にも必要です。 ◆ PARAMS では、ユーザー・イグジット・プロパティ・ファイルの名前を指定できます。
TABLE HR.*;	処理する表のリストを指定します。

図 2　　標準的な構成



ユーザー・イグジット・プロパティ

ユーザー・イグジットは、CUSEREXIT PARAMS で識別されるファイルからプロパティを読み取ります。デフォルトでは、ffwriter.properties から読み取ります。

プロパティ・ファイルには、ユーザー・イグジットによる操作方法の詳細が含まれています。各プロパティの詳細は、18 ページの「プロパティ」を参照してください。

推奨されるデータ統合方法

マイクロ・バッチ機能を最大限利用するには、データ統合ツールで次の処理を行う必要があります。

1. 制御ファイルを処理します。
2. 処理するファイルのリストを制御ファイルから読み取ります。
3. 制御ファイルの名前を変更します。
4. 制御ファイルから読み取ったカンマ区切りのファイルのリストを順に処理します。
5. 各データ・ファイルを処理し、完了したら、データ・ファイルを削除します。
6. 名前を変更した制御ファイルを削除します。

起動時、データ統合ツールは、名前が変更された制御ファイルをチェックし、以前に失敗した処理からリカバリする必要があるか確認します。

制御ファイルの名前が変更されている場合、最初のファイル・ロールオーバー時、ユーザー・イグジットは新規ファイルに書き込みます。これに、次のバッチ用のファイルのリストが含まれます。

ユーザー・イグジットがサマリー・ファイルを出力するようにも構成されている場合、オプションでデータ統合ツールはそのサマリー・ファイルも読み取り、処理したデータ・ファイルごとに処理した操作の数とサマリー・ファイル内のデータをクロスチェックできます。

データ・ファイルの生成

データ・ファイルは、ユーザー・イグジットのプロパティで*ライター*を構成することで生成されます。1 つのユーザー・イグジット・プロパティ・ファイルは複数のライターを持つことができるため、同じ入力データに対して複数の異なる形式の出力データ・ファイルを生成できます。

ライターは goldengate.flatfilewriter.writers プロパティに名前で追加されます。次に例を示します。

```
goldengate.flatfilewriter.writers=dsvwriter,diffswriter,binarywriter
```

プロパティ・ファイルの残りの部分には、指定された各ライターの詳細なプロパティが含まれます。プロパティの前にライター名を付けます。次に例を示します。

```
dsvwriter.files.onepertable=true
binarywriter.files.onepertable=false
binarywriter.files.oneperopcode=true
```

各ライターは、すべてのデータを 1 つの (ローリング) データ・ファイルに出力することも、入力表または操作タイプごとに 1 つの (ローリング) データ・ファイルを生成することもできます。これは、前述の例にある files.onepertable および files.oneperopcode プロパティで制御されます。

各ライターによって書き込まれるデータの出力形式は 2 つあり、モード・プロパティによって制御されます。これは次のいずれかです。

- DSV: デリミタ区切りの値
- LDV: 長さ区切りの値

次に例を示します。

```
dsvwriter.mode=dsv
binarywriter.mode=ldv
```

データ・ファイルが初めてディスクに書き込まれる際、一時的な拡張子が付けられます。ファイルがロールオーバーの基準を満たすと、拡張子はロールされた拡張子に変わります。制御ファイルが使用されている場合、必要に応じて制御ファイルが作成され、最終的なファイル名が制御ファイル内のリストに追加されます。また、ファイル・レベルの統計サマリーが生成される場合、ファイルのロールオーバー時に作成されます。

出力ディレクトリ (データ・ファイルと制御ファイルで異なる)、一時的な拡張子、ロールされた拡張子、制御ファイルの拡張子および統計サマリーの拡張子はすべてプロパティを使用して構成できます。出力構成の詳細は、22 ページの「出力ファイルのプロパティ」を参照してください。

書き込まれる各データ・ファイルは、出力スタイルに応じた命名規則に従います。表ごとに 1 つのファイルに書き込むファイルの場合、次のように名前に表名が含まれます。

```
MY.TABLE_2007-08-03_11:30:00_data.dsv
```

すべてのデータを 1 つのファイルに書き込むファイルの場合、次のように名前に表名は含まれません。

```
output_2007-08-03_11:30:00_data.dsv
```

基本的なデータの内容以外に、データ消費の際に便利なように*メタデータ列*を出力データに追加できます。これには、スキーマ (所有者) と表情報、ソース・コミット・タイムスタンプ、Oracle GoldenGate 読取り位置などがあります。メタデータ列の詳細は、28 ページの「メタデータ列」を参照してください。

データ・ファイルの内容は、モード、入力データ、追加するメタデータ列 (追加する場合) を決める各種プロパティ、列名を含めるかどうか、ビフォア・イメージを含めるかどうかなどによって異なります。出力データを決定するすべてのプロパティの詳細は、26 ページの「データ内容のプロパティ」を参照してください。

Oracle GoldenGate フラット・ファイル・アダプタの使用

Oracle GoldenGate フラット・ファイル・アダプタの継続的操作には、ファイル・ロールオーバーの管理、システムのチューニングに役立つ総計の収集、プロセスの管理、エラーの処理およびシステムのトラブルシューティングなどがあります。

制御ファイルの使用

制御ファイルには、ロールオーバーしたデータ・ファイルに関する情報が格納されます。制御ファイルが存在する場合、そのファイルに追加されます。存在しない場合、作成されます。すべてのデータを 1 つのファイルに出力するライターの場合、1 つの制御ファイルが作成されます。ライターが表ごと、または操作タイプごとに 1 つのファイルに出力する場合、制御ファイルも表ごと、または操作タイプごとに作成されます。

制御ファイルの生成、その出力ディレクトリ、接頭辞および拡張子は、22 ページの「出力ファイルのプロパティ」で定義されているプロパティによって制御されます。

各制御ファイルには、制御ファイルの作成以降ロールオーバーされたデータ・ファイルのカンマ区切りのリストが含まれます。ファイルはロールオーバーされた順にリストされます。これによって、データ統合ツールは、データ・ファイルが正しい順序で読み取られることと、すべて消費されたことを確認できます。

統計サマリーの使用

データ生成プロセスに関するサマリー統計を収集できます。この統計サマリー情報は、Oracle GoldenGate レポート・ファイルまたは個々のサマリー・ファイルに書き込まれます。

レポート・ファイルに書き込む場合、ユーザーは、この情報をファイルのロールオーバー時に書き込むか、期間に応じて定期的に書き込むかを決定できます。レポート・ファイルに書き込まれる情報は標準形式の出力で、レコード合計、各データベース操作タイプの合計、最後のレポート以降のデルタ、速度情報および各表の詳細が含まれます。

個々のサマリー・ファイルに書き込む場合、ファイルはロールオーバー・ファイルごとに作成されます。ロールオーバー・ファイルの統計情報は、デリミタで区切ってリストされます。サマリー・ファイルの拡張子、出力されるデータ、データ・デリミタおよび行デリミタはすべて制御できます。

35 ページの「統計およびレポート」に、統計ファイルとサマリー・ファイルのプロパティの詳細が記載されています。

Oracle GoldenGate プロセスの管理

標準的なデータ統合ソリューションに関係するプロセスは次のとおりです。

- ソース・データベースからトランザクションのデータを取得するプライマリ Extract プロセス
- 取得されたトランザクションのデータをソース・データベース・マシンからデータ統合マシンへネットワークを経由して移動する PASSTHRU データ・ポンプ Extract
- ユーザー・イグジットを実行するよう構成されている配信データ・ポンプ Extract

通常、元の取得と PASSTHRU データ・ポンプが含まれる Oracle GoldenGate インストールと、配信データ・ポンプが含まれるインストールは異なります。これらのインストールの両方ともで、Oracle GoldenGate Manager プロセスが稼働している必要があります。

これらのインストール内のプロセスは、Oracle GoldenGate GGSCI コマンドラインで起動や停止などの簡単なコマンドを使用して管理されます。これらのプロセスとその構成の管理の詳細は、*Oracle GoldenGate 管理者ガイド*を参照してください。

証跡リカバリ・モード

RECOVERYOPTIONS Extract パラメータによって、証跡への書き込み時に異常終了した Extract の再起動の動作が決まります。APPENDMODE は、リリース 10 以上の証跡のデフォルトです。異常終了した Extract は、追加モードで再起動する場合、リカバリ・マーカーと中断されたトランザクション全体を証跡に書き込みます。

Oracle GoldenGate フラット・ファイル・アダプタ・ファイル・ライターがこの証跡を読み取ると、部分的なトランザクションと、部分的なトランザクションを破棄する必要があることを示すリカバリ・マーカーを受信します。ファイル・ライターは、自身の出力ファイル内の位置を部分的なトランザクションの先頭に移動し、証跡ファイルの次のトランザクションで上書きします。

エラー・メッセージの特定

フラット・ファイル用 Oracle GoldenGate の操作時発生する可能性のあるエラーには 3 つのタイプがあります。

1. ユーザー・イグジットを実行する Extract プロセスが起動しません。
2. プロセスは起動しますが、その後異常終了します。
3. プロセスは正常に稼働しますが、データが正しくないか、存在しません。

最初の 2 つのケースでは、エラー・メッセージを検索する場所が多数あります。

- 標準の ggserr.log ファイル。Oracle GoldenGate プロセスの基本的な情報、実行履歴とエラー・メッセージ (エラーが発生した場合) が含まれます。
- dirrpt サブディレクトリにある、ユーザー・イグジットを実行する Extract プロセスの Oracle GoldenGate レポート・ファイル。たとえば、プロセス名が `ffwriter` の場合、レポート・ファイルは ffwriter.rpt です。これには、エラーに関するより詳細な情報が含まれる場合 (特に、ユーザー・イグジットではなく Oracle GoldenGate コア製品の問題の場合) があります。
- ユーザー・イグジット・ログ・ファイル内。名前は `log.logname` プロパティによって異なります。このファイルが存在しない場合、ユーザー・イグジットが起動していない可能性が高く、レポート・ファイルがその問題の切離しに役立ちます。

16 ページの「トラブルシューティング」に、エラー処理の詳細が含まれています。

トラブルシューティング

フラット・ファイル用 Oracle GoldenGate に関連する特定の問題について確認する前に、Oracle GoldenGate が正しく構成され、標準的な Oracle GoldenGate エラーが解決されていることを確認します。詳細は、『*Oracle GoldenGate トラブルシューティングおよびパフォーマンス・チューニング・ガイド*』を参照してください。

また、次の点も確認してください。

- Extract パラメータ・ファイル内の共有ライブラリ (.so または .dll) は正しいですか。パスが指定され、アクセス可能ですか。
- Extract パラメータ・ファイルで正しい `SOURCEDEFS` ファイルが指定されていますか。指定されたパスにあり、アクセス可能ですか。
- `SOURCEDEFS` ファイルにすべての必要な表が含まれていますか。
- `ffwriter.properties` ユーザー・イグジット・プロパティ・ファイルは Oracle GoldenGate インストール・ディレクトリにありますか。また、GG_USEREXIT_PROPFILE 環境変数で正しい名前とパスが指定されていますか。
- ユーザー・イグジット・プロパティ・ファイルで指定された出力ディレクトリは存在しますか。
- ファイル権限はそのディレクトリへの書込みに適していますか。

ユーザー・イグジット・ログ・ファイル (`logname_yyyymmdd.log`) を確認してください。

- ユーザー・イグジット・プロパティ・ファイルは正常に解析されますか。無効なプロパティがログ・ファイルに含まれていますか。

- その他のエラーまたは警告がログにありますか。

問題がまだ解決されない場合、次のようにします。

- log.level=DEBUG に設定します。
- 再起動し、ログ・ファイルを保存します。

Oracle サポートに連絡する前に、ログ・ファイル、ソース証跡ファイル、ソース定義ファイル、ユーザー・イグジット・プロパティ・ファイルおよび Extract パラメータ・ファイルを、書き込まれたデータ・ファイルとともに送信する準備をしてください。

第 4 章
プロパティ

プロパティの使用

プロパティ・ファイル内のプロパティにはすべて次の形式が使用されます。

```
fully.qualified.name=value
```

値は、1 つまたはカンマ区切りの文字列、整数またはブール値です。行の先頭に # 接頭辞を付けることでコメントをプロパティ・ファイルに入力できます。次に例を示します。

```
# This is a property comment
some.property=value
```

プロパティ自体もコメントにできます。これによって、以前のプロパティ設定を残したまま構成をテストできます。

ユーザー・イグジット・プロパティ

ユーザー・イグジットのプロパティには、ロギングを制御するプロパティ、名前やトランザクションの処理を制御する一般プロパティなどがあります。

ロギング・プロパティ

ロギングは次のプロパティによって制御されます。

log.logname

ログ・ファイル名の接頭辞を指定します。これは有効な ASCII 文字列である必要があります。ログ・ファイル名には、yyyymmdd 形式の現在の日付と .log 拡張子が付加されます。

次の例では、writer_20100803.log という名前のログ・ファイルが 2010 年 8 月 3 日に作成されます。ログ・ファイルは、プロセスの停止と起動に関係なく、毎日ロールオーバーします。

```
# log file prefix
log.logname=writer
```

次の例では、msgv_20100803.log という名前のログ・ファイルが 2010 年 8 月 3 日に作成されます。

```
# log file prefix
log.logname=msgv
```

log.level

すべてのモジュールを対象とする全体的なログ・レベルを指定します。構文は次のとおりです。

```
log.level=ERROR | WARN | INFO| DEBUG
```

ログ・レベルは次のように定義されています。

ERROR: エラーが発生した場合のメッセージのみ書き込ます。

WARN: エラーおよび警告メッセージを書き込みます。

INFO: エラー、警告および情報メッセージを書き込みます。

DEBUG: デバッグ・メッセージを含むすべてのメッセージを書き込みます。

デフォルトのロギング・レベルは、INFO です。この場合、メッセージは、起動時、停止時および操作中に定期的に生成されます。たとえば、次の例ではグローバル・ロギング・レベルを INFO に設定します。

```
# global logging level
log.level=INFO
```

注意 **レベルを DEBUG に切り替えると、メッセージが大量に生成され、パフォーマンスに影響する場合があります。**

log.tostdout

標準出力にログ情報が書き込まれるかどうかを制御します。この設定は、コマンドラインから起動された VAM と Extract プロセスが連携している場合、または stdout がレポート・ファイルに設定されているオペレーティング・システムで Extract プロセスが実行されている場合に有用です。ただし、Oracle GoldenGate プロセスは通常バックグラウンド・プロセスとして実行されます。

構文は次のとおりです。

```
goldengate.log.tostdout=true|false
```

デフォルトは、false です。

log.tofile

指定されたログ・ファイルにログ情報が書き込まれるかどうかを制御します。構文は次のとおりです。

```
log.tofile=true|false
```

デフォルトは、false です。true に設定されている場合、ログ出力は指定されたログ・ファイルに書き込まれます。

log.modules、*log.level.{module}*

ユーザー・イグジットを構成する各ソース・モジュールのログ・レベルを指定します。これは通常詳細デバッグの場合にのみ使用されます。ロギング・レベルをモジュールごとに DEBUG に引き上げ、問題のトラブルシューティングに役立てることができます。Oracle サポートから依頼されないかぎり、デフォルトのレベルは変更しないでください。

一般プロパティ

一般プロパティは、ファイル・ライター名、チェックポイント、トランザクションの処理、タイムスタンプの表示および列とオブジェクト名に使用される形式を制御します。

goldengate.flatfilewriter.writers

ユーザー・イグジット内で実行されるライターの名前を指定します。複数の文字列値を入力すると、同一ユーザー・イグジット内で複数の名前のライターを実行できます。次に例を示します。

```
goldengate.flatfilewriter.writers=dsvwriter,diffswriter,binwriter
```

等号記号またはカンマの前後に空白がないようにしてください。ファイル内の他のすべてのプロパティにライター名を接頭辞として付けます。

goldengate.userexit.buffertxs

出力の前にトランザクション全体が読み取られるかどうかを制御します。`true` に設定されている場合、出力の前にトランザクション全体が証跡から読み取られます。次に例を示します。

```
goldengate.userexit.buffertxs=true
```

デフォルトは、`false` です。これを true に設定することは、`numops` メタデータ列が使用される場合にのみ有用です。現在、`numops` 値を算出する唯一の方法は、トランザクションをバッファし、一度に 1 つのトランザクションを出力することです。

goldengate.userexit.chkptprefix

チェックポイント・ファイル名に追加される接頭辞の文字列値を指定します。複数のデータ・ポンプを実行する場合、チェックポイント接頭辞をプロセスの名前に設定します。次に例を示します。

```
goldengate.userexit.chkptprefix=pump1_
```

goldengate.userexit.chkpt.ontxend

ファイルをロールオーバーする必要があるかをトランザクションごとにチェックするか、Extract プロセスのチェックポイント時にのみチェックするかを制御します。`true` に設定されている場合、アダプタは、トランザクションの処理後、ファイルをロールオーバーする必要があるかをチェックします。必要がある場合、ロールオーバーが実行され、チェックポイント・ファイルが更新されます。これは、出力ファイルの内容を厳格に管理する必要がある場合に有用です。たとえば、午前 0 時にロールオーバーする前に午前 0 時までのすべてのデータをファイルに書き込む必要がある場合、トランザクションごとにチェックが行われることが重要です。次に例を示します。

```
goldengate.userexit.chkpt.ontxend=true
```

デフォルトは、`false` です。`false` に設定されている場合、アダプタは、Extract のチェックポイント時 (デフォルトでは 10 秒ごと) にロールオーバーについてチェックします。

goldengate.userexit.datetime.removecolon

日付と時間の間にコロンが書き込まれるかどうかを制御します。`false` に設定されている場合、日付と時間の列値は、Oracle GoldenGate 証跡のデフォルト形式 `YYYY-MM-DD:HH:MI:SS.FFFF` で出力ファイルに書き込まれます。`true` に設定されている場合、日付と時間の間にコロンがない `YYYY-MM-DD HH:MI:SS.FFF` に形式が変更されます。デフォルトは、`false` です。

```
goldengate.userexit.datetime.removecolon=true
```

goldengate.userexit.timestamp

レコードのタイムスタンプをローカル時間で出力するか、協定世界時 (UTC) で出力するかを制御します。これが `utc` に設定されていない場合、レコードのタイムスタンプはローカル・タイムゾーンを使用

してローカル時間で出力されます。デフォルトは、ローカル時間です。

```
goldengate.userexit.timestamp=utc
```

goldengate.userexit.datetime.maxlen

日時列の出力最大長を制御します。これを整数値に設定すると、列値がその長さに切り捨てられます。日時の形式は `YYYY-MM-DD:HH:MI:SS.F(9)` のため、日時列の最大長は 29 文字です。

次に例を示します。

```
goldengate.userexit.datetime.maxlen=19
```

`goldengate.userexit.maxlen=19` に設定すると、秒の端数のない日時に切り捨てられます。`goldengate.userexit.maxlen=10` に設定すると、日付のみに切り捨てられます。デフォルトは、完全な日時列値を出力することです。

goldengate.userexit.utf8mode

列データ、表名、ファイル名および列名が UTF8 文字セットで返されるかどうかを制御します。これが `false` に設定されている場合、すべてのデータはオペレーティング・システムの文字セットになります。デフォルトは、`false` です。

構文は次のとおりです。

```
goldengate.userexit.utf8mode=true|false
```

ファイル・ライター・プロパティ

ファイル・ライター・プロパティは、出力ファイルの形式およびファイルの書込み方法を制御します。

出力形式のプロパティ

次のプロパティは、値のデリミタ・タイプと列のグループ化を設定します。

mode

出力形式が DSV か LDV かを制御します。

- **DSV**: デリミタ区切りの値。次に例を示します。

  ```
  POSITION«OPCODE«TIMESTAMP«COLVALA«COLVALB«ETC
  ```

  注意 これはカンマ区切りの値 (CSV) に限定されません。デリミタはカンマである必要はありません。

- **LDV**: 長さ区切りの値。次に例を示します。

  ```
  0109TIMESTAMPI302MY05TABLEP042000P03ETC
  ```

  注意 長さは ASCII またはバイナリです。一部のメタデータ列は固定長で (28 ページの「メタデータ列」を参照)、この形式では Unicode マルチバイト・データがサポートされます。

  次に例を示します。

  ```
  dsvwriter.mode=dsv
  binarywriter.mode=ldv
  ```

注意 下位互換のために、dsv のかわりに csv が、ldv のかわりにバイナリが許容されます。かわりのオプションを使用する場合、出力形式に違いはありません。

groupcols

列名、ビフォア値およびアフター値をグループ化するかどうかを制御します。

構文は次のとおりです。

```
{writer}.groupcols=true|false
```

デフォルトは、false です。これによって、COL1 と COL2 の次の例に示すように、名前、ビフォア値およびアフター値がセットとして一緒にリストされます。

```
"COL1", COL1_B4, COL1, "COL2", COL2_B4, COL2
```

プロパティが true に設定されている場合、列は、すべての名前のセット、すべてのビフォア値のセットおよびすべてのアフター値のセットにグループ化されます。

```
"COL1", "COL2", COL1_B4, COL2_B4, COL1, COL2
```

出力ファイルのプロパティ

次のプロパティは、ファイルの書込み方法、書込み先とその拡張子を制御します。これは、ライター・モードとデータの内容に依存しません。

files.onepertable

データが複数のローリング・ファイル (入力データ内の表ごとに 1 つ) に分割されるか、すべてのデータが 1 つのローリング・ファイルに書き込まれるかを制御します。デフォルトは、true です。

構文は次のとおりです。

{*writer*}.files.onepertable=true|false

次の例では、dvswriter は表ごとに 1 つのファイルを作成し、binarywriter はすべてのデータを 1 つのファイルに書き込みます。

```
dsvwriter.files.onepertable=true
binarywriter.files.onepertable=false
```

files.oneperopcode

挿入、更新、削除または主キー操作コードに基づいてデータが分割されるかどうか制御されます。

たとえば、次の設定では、挿入、更新、削除および主キーの更新に対して個別の出力ファイルが作成されます。

```
dvswriter.files.oneperopcode=true
```

デフォルトは false で、操作のタイプに関係なく、すべてのレコードが同一ファイルに出力されます。

このプロパティ以外に files.formatstring プロパティも変更し、%O プレースホルダーを受け入れる必要があります。files.oneperopcode プロパティが設定されている場合、これは、ファイル名が作成される際に操作コードが書き込まれる位置を示します。そのプロパティが設定されている場合、デフォルトのファイル名にも操作コードが含まれます。

files.prefix

データ・ファイルおよび制御ファイルの接頭辞として使用される値を指定します。このプロパティは、ライターが表ごとに 1 つのモード (`files.onepertable=true`) でない場合にのみ適用されます。データ・ファイルでは、`files.formatstring` プロパティが使用されている場合、接頭辞は無視されます。

デフォルトでは、接頭辞は文字列 `output` に設定されます。`data1` という名前のファイルは、デフォルトで `outputdata1` になります。次の例を使用すると、ファイル名は `test_data1` になります。

```
dsvwriter.files.prefix=test_
```

files.data.rootdir、*files.data.ext*、*files.data.tmpext*

すべてのデータ・ファイルの場所と拡張子を指定します。ロールオーバー前のファイルの拡張子は `tmpext` で、ロールオーバー後は `ext` です。拡張子は `ext` のみである必要はありません。ファイル名の拡張子の前に文字を追加してデータ出力を区別できます。指定された出力ディレクトリが存在すること、Oracle GoldenGate プロセスを実行しているユーザーに、そのディレクトリへの適切な書込み権限があることを確認してください。次に例を示します。

```
# specify the root directory for outputting data files
dsvwriter.files.data.rootdir=./out

# determine the extension for data files when rolled over
dsvwriter.files.data.ext=_data.dsv

# determine the extension for data files before rolling over
dsvwriter.files.data.tmpext=_data.dsv.temp
```

files.control.use、*files.control.rootdir*、*files.control.ext*

`files.control.use` は true または false のブール値で、デフォルトは true です。その他は ASCII 値です。これらのプロパティによって、制御ファイルのユーザー、場所および拡張子が決まります。制御ファイルは、関係するデータ・ファイルと同じ名前の接頭辞を共有しますが、定義された拡張子を持ちます。デフォルトでは、`files.control.ext` は `.control` です。次に例を示します。

```
# specify whether or not to output a control file
dsvwriter.files.control.use=true

# specify the extension to use for control files
dsvwriter.files.control.ext=_data.control

# directory in which to place control files, defaults to data directory
dsvwriter.files.control.rootdir=./out
```

files.control.delim.chars/code、*files.control.eof.chars/code*

データ・デリミタまたは行末インジケータとして使用される値を文字または 16 進コードで指定します。デリミタのデフォルトはカンマ (,) です。デフォルトの新規行トリガーは、プラットフォームで有効な `newline` 文字です。

たとえば、カンマをデータ・デリミタとしてオーバーライドするには、次のようにします。

```
dvswriter.files.control.delim.chars=#
```

たとえば、新規行インジケータを設定するには、次のようにします。

```
dsvwriter.files.control.eol.chars=\n
```

files.formatstring

データ・ファイルのファイル名の作成時に使用されるファイル名フォーマット文字列を指定します。フォーマット文字列は `files.prefix` プロパティをオーバーライドします。ファイル名フォーマット文字列の構文は、次のプレースホルダがファイル名に追加可能な点以外、標準の C のフォーマットの構文と似ています。

%s = schema

%t = table

%n = seqno

%d = timestamp

%o = opcode

`seqno` の形式を指定できます。たとえば、%05n は、5 桁が表示され、0 でパディングされます。`seqno` は 0 から始まり、ファイルがロールオーバーするたびに 1 ずつ増分されます。`long int` として格納されるため、最大値はプラットフォームによって異なります。たとえば、64 ビット・マシンの場合、最大値は 2^64-1 です。

これらのプレースホルダは、ユーザーが指定するテキストと組み合せて任意の順序で使用できます。次に例を示します。

```
dsvwriter.files.formatstring=myext_%d_%010n_%s_%
```

files.data.bom.code

ファイルの先頭に書き込まれるバイト順マーカー (BOM) である 16 進数の値を指定します。BOM がファイル内のデータと合致していることを確認する必要があります。16 進値が指定されていない場合、マーカーは書き込まれません。

次の例では、すべての出力ファイルの最初のバイトとして UTF8 BOM `efbbf` が書き込まれます。

```
dsvwriter.files.data.bom.code=efbbbf
```

files.includeprocess.name

Extract プロセスの名前がファイル名の一部として含められるかどうかを制御します。デフォルトは、false です。

構文は次のとおりです。

```
files.includeprocess.name=true|false
```

files.useownerfiles

ファイルを所有する Extract プロセスを識別するための特別なファイルが作成されるかどうかを制御します。デフォルトは、false です。

構文は次のとおりです。

```
files.useownerfiles=true|false
```

ファイル・ロールオーバー・プロパティ

次のプロパティによって、ファイルのロールオーバーのポリシーが決まります。

files.data.rollover.time

最初のレコードがファイルに書き込まれてからファイルがロールオーバーされるまでの経過時間の最大秒数を指定します。次に例を示します。

```
# number of seconds before rolling over
dsvwriter.files.data.rollover.time=10
```

files.data.rollover.size

ファイルがロールオーバーされるまでにファイルに書き込まれる最大バイト数を指定します。

この例では、最大値を 10,000 バイトに設定します。

```
# max file size in KB before rolling over
dsvwriter.files.data.rollover.size=10000
```

files.data.norecords.timeout

データがファイルに書き込まれてファイルがロールオーバーされるまで待機する経過時間の最大秒数を指定します。デフォルトは、120 秒です。

この例では、タイムアウト間隔を 10 秒に設定します。

```
# roll over in case no records for a period of time
dsvwriter.files.data.norecords.timeout=10
```

files.rolloveronshutdown

Extract プロセスの停止時のロールオーバーのポリシーを制御します。この値が false の場合、空の一時ファイルはすべて削除されますが、データが含まれたファイルは一時ファイルのまま残されます。このプロパティが true の場合、空でない一時ファイルはすべて、ロールされたファイル名にロールオーバーされ、チェックポイントが書き込まれて、空の一時ファイルが削除されます。次に例を示します。

```
# roll over non-empty and delete all empty files when Extract stops
binarywriter.files.rolloveronshutdown=true
```

> **注意** **時間やサイズを使用できます。両方を使用する場合、先に達した方でロールオーバーが発生します。タイムアウト間隔によって、処理されるレコードがない場合でも、データが含まれているファイルはロールオーバーされることが保証されます。時間もサイズも指定されていない場合、最大サイズのデフォルトである 1MB を超えると、ファイルはロールオーバーされます。**

files.data.rollover.timetype

ファイル・ロールオーバーのトリガーにシステム時間ではなく、ユリウスのコミット・タイムスタンプを使用するかどうかを制御します。構文は次のとおりです。

```
{writer}.files.data.rollover.timetype=commit|system
```

次の例では、ソース証跡レコードのコミット・タイムスタンプを使用してロールオーバーを決定します。

```
dsvwriter.files.data.rollover.timetype=commit
```

デフォルトでは、システム時間を使用していつファイルをロールオーバーするかを決定します。

files.data.rollover.multiple

最初にレコードを受信した時間に関係なく、すべてのファイルを同時にロールオーバーするかどうかを制御します。通常、ファイルは時間またはサイズのプロパティに基づいて個々にロールオーバーされ

ます。時間はロールオーバー期間に基づくため、レコードが特定のファイルに最初に書き込まれた時間に依存します。場合によっては (特に、表ごとに 1 つのファイルにデータを出力する場合)、データが最初にファイルに書き込まれた時間に関係なく、現在開いているすべてのファイルを同時にロールオーバーすることがあります。

次の例では、すべてのファイルを同時にロールオーバーするようアダプタに指定します。

```
dsvwriter.files.data.rollover.multiple=true
```

デフォルト値は、`false` です。

files.data.rollover.attime

アダプタがファイルをロールオーバーする時間を指定します。指定された時間を 24 時間形式 (`HH:MM`) で入力します。1 つの値の入力のみサポートされます。ワイルドカード (*) は、時間に対してのみサポートされます。構文は次のとおりです。

```
{writer}.files.data.rollover.attime={time_specifier}
```

次の例では、毎正時に新しいファイルにロールオーバーします。

```
dsvwriter.files.data.rollover.attime=*:00
```

次の例では、毎時 15 分にロールオーバーします。

```
dsvwriter.files.data.rollover.attime=*:15
```

`rollover.timetype` プロパティによって、使用される時間がシステム時間かコミット時間かが決まることに注意してください。

csvwriter.writebuffer.size

書込みバッファ・チャンク・サイズを指定します。システム書込みコール数の削減に使用します。次に例を示します。

```
csvwriter.writebuffer.size=36863
```

データ内容のプロパティ

次のプロパティによって、データ・ファイルに書き込まれるデータが決まります。これらのプロパティは、出力データの形式に依存しません。

rawchars

文字データを元のバイナリ形式のままにするか、ASCII として出力するかを制御します。デフォルトは、false です。ASCII に変換しない Unicode マルチバイト・データが入力データに含まれている場合、このプロパティを設定します。次に例を示します。

```
# whether to output characters as ASCII or binary (for Unicode data)
dsvwriter.rawchars=false
binarywriter.rawchars=true
```

includebefores

データのビフォア・イメージとアフター・イメージの両方が更新操作の出力に含まれるかどうかを制御します。デフォルトは、false です。これは、ビフォア・イメージが元のデータで提供されていて、`getupdatebefores` が処理チェーンのすべての Oracle GoldenGate パラメータ・ファイルに含まれてい

る場合にのみ関係します。次に例を示します。

```
# whether to output update before images
dsvwriter.includebefores=true
```

これによって、. . ."VAL_BEFORE_1","VAL_1","VAL_BEFORE_2","VAL_2". . .が生成されます。

afterfirst

includebefores が true に設定されている場合に、アフター・イメージがビフォア・イメージの前に書き込まれるかどうかを制御します。

次に例を示します。

```
dsvwriter.afterfirst=true
```

この true の設定によって、アフター・イメージがビフォア・イメージの前にリストされます。

```
"VAL_1", "VAL_BEFORE_1", "VAL_2", "VAL_BEFORE_2"
```

デフォルトは、false です。この場合、アフター・イメージはビフォア・イメージの後に書き込まれます。

includecolnames

列値の前に列名が出力されるかどうかを制御します。デフォルトは、false です。次に例を示します。

```
# whether to output column names
dsvwriter.includecolnames=true
```

これによって、"COL_1","VAL_1","COL_2","VAL_2" が生成されます。

omitvalues

出力ファイルで列値が省略されるかどうかを制御します。デフォルトは、false です。次に例を示します。

```
# whether to output column values
dsvwriter.omitvalues=false
```

includecolnames も true に設定されている場合、これによって "COL_1","COL_2" が生成されます。

diffsonly

すべての列が出力されるか、ビフォア・イメージがアフター・イメージと異なるもののみ出力されるかを制御します。デフォルトは、false です。これは更新にのみ適用され、処理チェーンのすべての Oracle GoldenGate パラメータ・ファイルに GETUPDATEBEFORES が必要です。このプロパティは、includebefores プロパティに依存しません。次に例を示します。

```
# whether to output only columns with differences between before and
# after images (deletes and inserts have all available columns)
dsvwriter.diffsonly=true
```

これによって、. . ."VAL_1",,,"VAL_4",,,"VAL_7". . .が生成されます。

omitplaceholders

欠落している列のデリミタ / 長さが出力に含まれるかどうかを制御します。デフォルトは、false です。これは、COMPRESSUPDATES または COMPRESSDELETES フラグが処理チェーンの Oracle GoldenGate パラ

メータ・ファイルに含まれていた場合の更新および削除に適用されます。この場合、値が欠落している場合があります。また、diffsonly が true の場合、差異のない値も欠落していることになります。次に例を示します。

```
# whether to skip record delimiters if columns are missing
dsvwriter.omitplaceholders=true
```

これによって、. . ."VAL_1",,,"VAL_4",,,"VAL_7". . .が次のように変更されます。

```
. . ."VAL_1","VAL_4","VAL_7". . .
```

メタデータ列

メタデータ列は、実際のレコード・データではなく、レコードに関するデータを含むオプションの Extract 列です。これらの列は、出力レコードの先頭、すべての列値の前に書き込まれます。

有効なメタデータ列

有効なメタデータ列は次のとおりです。

position: 証跡内のレコードの一意の位置インジケータ。

opcode: 挿入、更新、削除、主キー更新の各レコードに対してそれぞれ、I、U、D、K。

txind: トランザクション内の通常のレコードの位置 (0: 先頭、1: 中間、2: 末尾、3: 専用)。

txoppos: 0 から始まるトランザクション内のレコードの位置。

schema: 変更レコードのスキーマ (所有者) 名。

table: 変更レコードの表名。

schemaandtable: スキーマ名と表名が schema.table として連結されます。

timestamp: レコードのコミット・タイムスタンプ。

@<token name> : Extract パラメータ・ファイルに定義されているトークン値。

$getenv: *Oracle GoldenGate リファレンス・ガイド*で説明されている GETENV 値。たとえば、$GGHEADER.OPCODE です。

%COLNAME: データ列の値。

numops: 現在のトランザクションの操作の数。goldengate.userexit.buffertxs が true でない場合、この値は常に 1 です。

numcols: 出力される列の数。この値は、このメタデータ列が使用される時点までにメタデータ列として出力された列数を元のレコードの列数から引いた数に等しくなります。

"<value>": 任意のリテラル値。

メタデータ列の使用

メタデータ列を使用する場合、検討の必要な事項があります。

- opcode および txind に対する ASCII 値はオーバーライドできます。
- LDV の場合、メタデータ列は可変または固定長です。

- position は、16 進数または 10 進数で書き込むことができます。
- メタデータ列は内部の値でも、元のデータの列から読み取ることもできます。
- リテラル値は、引用符で囲んで表します。リテラル値が指定される場合、その値は、指定されたメタデータ列の位置に適切な引用符ポリシーを使用して文字列として出力されます。
- 列値は %COLNAME で表します。列値が指定されている場合、列値は、出力レコードの列値セクションではなくメタデータ・セクションに出力されます。これを使用して、出力される表に関係なく、列が常にレコード内の同じ位置に出力されるようにすることができます。

次のプロパティがメタデータ列に適用されます。

metacols

出力するメタデータ列を出力順に指定します。ASCII 値の複数の名前をカンマで区切って入力します。次に例を示します。

```
# which metacols to output and in which order
dsvwriter.metacols=timestamp,opcode,txind,position,schema,table
```

metacols.{metacol-name}.fixedlen

メタデータ列に対して書き込まれるデータの長さを決める整数値を指定します。実際のデータが固定長より長い場合、切り捨てられます。短い場合、出力はパディングされます。次に例を示します。

```
# timestamp is fixed length
dsvwriter.metacols.timestamp.fixedlen=23
```

これによって、2011-08-03 10:30:51.123456 が 2011-08-03 10:30:51.123 に切り捨てられます。

metacols.{metacol-name}.column

内部の値を使用するかわりにメタデータ列のデータ値の列名として使用する ASCII 値を指定します。設定される場合、この列名は、ユーザー・イグジットで処理されるすべての表に存在する必要があります。この列名を表ごとにオーバーライドする方法は現在ありません。たとえば、列の内部タイムスタンプをオーバーライドするには、次のようにします。

```
# timestamp is read from a column
dsvwriter.metacols.timestamp.column=MY_TIMESTAMP_COL
```

metacols.{token-name}.novalue.chars/code

必要なトークン値が使用できない場合に使用される文字または 16 進コードを表す値を指定します。chars の場合、ASCII 値を使用し、code の場合、16 進値を使用します。デフォルト値は、NO VALUE です。次に例を示します。

```
dsvwriter.metacols.TKN-SCN.novalue.chars=0
```

metacols.{metacol-name}.fixedjustify

メタデータ列値の位置合せが左か右かを制御します。デフォルトでは、すべてのメタデータ列は左詰めです。たとえば、トークンを右詰めにするには、次のようにします。

```
dsvwriter.metacols.TKN-SCN.fixedjustify=right
```

metacols.{metacol-name}.fixedpadchar.chars/code

メタデータ列のパディングに使用される文字またはコード値を指定します。chars の場合、ASCII 値を使用し、code の場合、16 進値を使用します。パディングに使用されるデフォルトの文字は空白 (" ") です。次に例を示します。

```
dsvwriter.metacols.TKN-SCN.fixedpadchar.chars=0
```

metacols.opcode.insert.chars/code、*metacols.opcode.update.chars/code*、*metacols.opcode.delete.chars/code*

挿入、更新および削除操作を識別するデフォルトの文字のオーバーライド値を指定します。chars の場合、ASCII 値を使用し、code の場合、16 進値を使用します。デフォルト値は、挿入の場合 I、更新の場合 U、削除の場合 D です。

次の例では、挿入の場合 INS、更新の場合 UPD、削除の場合 DEL を使用するようアダプタに指定します。

```
# op code values are overridden
dsvwriter.metacols.opcode.insert.chars=INS
dsvwriter.metacols.opcode.update.chars=UPD
dsvwriter.metacols.opcode.delete.chars=DEL
```

metacols.txind.begin.chars/code、*metacols.txind.middle.chars/code*、*metacols.txind.end.chars/code*、*metacols.txind.whole.chars/code*

トランザクションの先頭、中間、末尾か、あるいはトランザクション全体かの識別に使用される値のオーバーライド値を指定します。chars の場合、ASCII 値を使用し、code の場合、16 進値を使用します。デフォルト値は、先頭の場合 0、中間の場合 1、末尾の場合 2、全体の場合 3 です。

次の例では、0、1、2、3 を文字 B、M、E および W でオーバーライドします。

```
# tx indicator values are overridden
dsvwriter.metacols.txind.begin.chars=B
dsvwriter.metacols.txind.middle.chars=M
dsvwriter.metacols.txind.end.chars=E
dsvwriter.metacols.txind.whole.chars=W
```

metacols.position.format

position メタデータ列の出力を 10 進形式で行うか、16 進形式で行うかを制御します。16 進の場合、通常 16 文字の値です。10 進の場合、長さは変わります。これには、Extract プロセスが読み取る Oracle GoldenGate 証跡の順序番号と RBA が現在含まれます。次に例を示します。

```
# position is in decimal format (seqno0000000rba)
dsvwriter.metacols.position.format=dec
```

seqno が 12、rba が 12345 の場合、これによって 120000012345 が生成されます。

```
binarywriter.metacols.position.format=hex
```

seqno が 12、rba が 12345 の場合、これによって 0000000c00003039 が生成されます。

metacols.{COLNAME}.omit

COLNAME 列をメタデータとして使用し、出力しないかどうかを制御します。

次の例では、numcols 列をメタデータとして使用するが、出力しないことを指定します。

```
dsvwriter.metacols.numcols.omit=true
```

begintx.metacols, endtx.metacols

トランザクションの先頭と末尾をマークするために使用するメタデータ列を指定します。これらのマーカー・レコードは、トランザクションを構成する操作レコードの前後に出力ファイルに (行末デリミタ付きで) 書き込まれます。

構文は次のとおりです。

```
{writer}.begintx.metacols={metacols list}
```

次の例では、トランザクションの先頭を文字 B とトランザクション内の操作の数でマークすることとを指定します。

```
dvswriter.begintx.metacols="B",numops
```

次の例では、トランザクションの末尾のマーカーは文字 E になります。

```
dvswriter.endtx.metacols="E"
```

既存のメタデータ列をトランザクションの先頭と末尾のマーカーに使用できます。レコードの列値または特定のプロパティ (表名など) を begintx.metacols に指定する場合、トランザクション内の最初のレコードの値が使用されます。endtx.metacols の場合、最後のレコードの値が使用されます。

たとえば、トランザクションに次のレコードがあるとします。

```
rec=0,table=tabA,operation=insert,col1=val1,col2=val2
rec=1,table=tabA,operation=update,col1=val3,col2=val4
rec=2,table=tabA,operation=delete,col1=val5,col2=val6
rec=3,table=tabB,operation=update,col1=val7,col2=val8
```

プロパティは次のように設定されています。

```
dvswriter.begintx.metacols="B",table,%col2
dvswriter.endtx.metacols="E",table,%col2
```

トランザクションの先頭のマーカーは "B","tabA","val2"、末尾のマーカーは "E","tabB","val8" になります。

先頭または末尾のマーカーに numops を使用してトランザクション内の操作の数を出力する場合、次の設定も必要です。

```
goldengate.userexit.buffertxs=true
```

注意 **このプロパティが設定されている場合、アダプタはトランザクションをメモリーにバッファするため、システムによって処理されるトランザクション内の操作の数を制限する必要があります。**

DSV 固有のプロパティ

DSV ファイルのレコード形式は次のようになります。

```
{[METACOL][FD]}n{[COL][FD]}m[LD]
```

条件 :

METACOL は、任意の定義済メタデータ列です。

COL は、任意のデータ列です。

FD は、フィールド・デリミタです。

LD は、行デリミタです。

列値は、"2007-01-10 10:20:31","U","MY.TABLE", 2000,"DAVE" のように引用符で囲むことができます。

dsv.nullindicator.chars/code

デリミタ区切りファイルで NULL 値に使用する文字を指定します。これらの値は、空の文字列のデフォルトである NULL 値をオーバーライドします。chars の場合、ASCII 値を使用し、code の場合、16 進値を使用します。次に例を示します。

```
dsvwriter.dsv.nullindicator.chars=NULL
dsvwriter.dsv.nullindicator.code=0a0a0a0a
```

dsv.fielddelim.chars/code

フィールド・デリミタのオーバーライド値を指定します。デフォルトは、カンマ (,) です。 chars の場合、ASCII 値を使用し、code の場合、16 進値を使用します。次に例を示します。

```
# define the characters to use for field delimiters in DSV files
dsvwriter.dsv.fielddelim.chars=«
```

dsv.linedelim.chars/code

行デリミタのオーバーライド値を指定します。デフォルトは、オペレーティング・システムに対応した改行です。chars の場合、ASCII 値を使用し、code の場合、16 進値を使用します。次に例を示します。

```
# define the characters to use for line delimiters in DSV files
dsvwriter.dsv.linedelim.chars=\n
```

dsv.quote.chars/code

引用符のオーバーライド値を指定します。デフォルトは、二重引用符 (") です。chars の場合、ASCII 値を使用し、code の場合、16 進値を使用します。次に例を示します。

```
# define the characters to use for quotes in DSV files
dsvwriter.dsv.quotes.chars='
```

dsv.quotes.policy

引用符を付けるためのポリシーを制御します。

構文は次のとおりです。

```
{writer}.dsv.quotes.policy={default|none|always|datatypes}
```

条件 : **default:** 日付と文字にのみ引用符を付けます。

never: メタデータ列にも列値にも引用符を付けません。

always: すべてのメタデータ列と列値に引用符を付けます。

datatypes: 特定のデータ型にのみ引用符を付けます。

このプロパティが設定されている場合、dsv.quotealways プロパティはオーバーライドされます。dsv.quotes.policy.datatypes プロパティを使用して引用符を付けるデータ型を指定します。

dsv.quotes.policy.datatypes

dsv.quotes.policy が datatype に設定されている場合に整数、文字、単精度浮動小数点または日時データ型に引用符が付けられるかどうかを制御します。構文は次のとおりです。

```
{writer}.dsv.quotes.policy.datatypes=[char][,integer][,float][,date]
```

たとえば、次の指定では、アダプタは文字値と日時値にのみ引用符を付けます。

```
dsvwrite.dsv.quotes.policy.datatypes=char,date
```

データ型が指定されない場合、データ型のデフォルトはすべてのデータ型で、always と同じです。

dsv.nullindicator/fielddelim/linedelim/quotes.escaped.chars/code

前述の値に対するエスケープ値を指定します。設定されている場合、エスケープされていない値がないかすべての値がチェックされ、出力時にエスケープされた値に置き換えられます。chars の場合、ASCII 値を使用し、code の場合、16 進値を使用します。次に例を示します。

```
# (optionally) you can define the characters (or code) to use
# to escape these values if found in data values
dsvwriter.dsv.quotes.escaped.chars=""
```

これによって、"some text" は ""some text"" に変更されます。

dsv.onecolperline

列値ごとに強制的に改行されるかどうかを制御します。各行に、このライター用に定義されたメタデータ列も含まれます。デフォルトは、false です。次に例を示します。

```
# Force each column onto a new line with its own meta cols
dsvwriter.dsv.onecolperline=true
```

これによって、{metacols},val_1,val_2 が次のように変更されます。

```
{metacols},val1
{metacols},val2
```

dsv.quotealways

数値の場合でも、各列が引用符で囲まれるかどうかを制御します。デフォルトは、false です。

注意 **このプロパティのかわりに dsv.quotes.policy が導入されています。このプロパティは下位互換のためにのみサポートされます。dsv.quotes policy が設定されている場合、dsv.quotealways に設定された値は無視されます。**

次に例を示します。

```
dsvwriter.dsv.quotealways=true
```

. . .,1234,"Hello",10 が . . ."1234","Hello","10" に変更されます。

LDV 固有のプロパティ

LDV ファイルのレコード形式は次のようになります。

```
[RECLEN][METACOLS]{[FLAG][LEN][VALUE]}n
```

条件 :

RECLEN は、フル・レコードの長さ (バイト) です。

METACOLS は、すべての選択されたメタデータ列です。

FLAG は、M(欠落している)、P(存在している) または N (null) です。

LEN は、列値の長さ (欠落または null の場合 0) です。

VALUE は、列値です。

次に例を示します。

```
01072007-01-10 10:20:31U302MY05TABLEP042000M00N00P04DAVE
```

ldv.vals.missing.chars/code、ldv.vals.present.chars/code、ldv.vals.null.chars/code

欠落、存在および null インジケータのオーバーライド値を指定します。chars の場合、ASCII 値を使用し、code の場合、16 進値を使用します。次に例を示します。

```
binarywriter.ldv.vals.missing.chars=MI
binarywriter.ldv.vals.present.chars=PR
binarywriter.ldv.vals.null.chars=NL
```

ldv.lengths.record.mode、ldv.lengths.field.mode

レコード長とフィールド長の出力モードを制御します。値は、binary または ASCII のいずれかです。デフォルトは、binary です。

binary の場合、ファイルに書き込まれる数値はバイナリ・バイトでエンコードされます。ASCII の場合、長さの 10 進値を表す文字が使用されます。次に例を示します。

```
binarywriter.ldv.lengths.record.mode=binary
binarywriter.ldv.lengths.field.mode=binary
```

ldv.lengths.record.length、ldv.lengths.field.length

レコード長およびフィールド長を整数値として指定します。これは、モードが ASCII の場合、使用される 10 進数の決められた数を表します。binary の場合、バイト数を表します。

ASCII モードでは長さは任意の値にできますが、長さが最大値を超えると、イグジットは停止します。バイナリ・モードでは、長さは 2、4 または 8 にできますが、レコード長は固定長より大きい値である必要があります。次に例を示します。

```
# Lengths can be binary (2,4, or 8 bytes) or ASCII (any length)
binarywriter.ldv.lengths.record.length=4
binarywriter.ldv.lengths.field.length=2
```

統計およびレポート

データ・ファイルに書き込まれるデータに関する統計を取得する方法は 2 つあります。

- Oracle GoldenGate レポート・ファイルに書き込まれるレポートとして
- ロールオーバー時にデータ・ファイルに関連付けられる個別のサマリー・ファイルとして

これらの 2 つのメカニズムを組み合せて、あるいは別々に使用できます。

取得可能なデータには、1) 処理されたレコードの総数とこれを挿入、更新、削除に分類したもの、2) 表ごとの処理されたレコード数とこれを分類したもの、3) 全体的な速度および表ごとの速度、4) 最後のレポート以降のこれらのデルタ、などがあります。レポートは時間ベースにも、ファイルのロールオーバーに同期させることもできます。

このデータはレポート・ファイルまたはロールオーバー時にデータ・ファイルにリンクされるサマリー・ファイルに書き込まれます。レポートの形式は固定です。サマリー・ファイルにはデータが区切られた形式で含まれますが、特定のデータ・ファイルの内容に関するものが含まれます。これはデータ統合製品によって使用され、処理がクロスチェックされます。データ・ファイルと同じ名前が付けられますが、拡張子は異なります。

statistics.toreportfile

統計が Oracle GoldenGate レポート・ファイルに出力されるかどうかを制御します。次に例を示します。

```
binarywriter.statistics.tosummaryfile=true
```

statistics.period

統計の期間を指定します。値は、`timebased` または `onrollover` のいずれかです。

次に例を示します。

```
dsvwriter.statistics.period=onrollover
dsvwriter.statistics.period=timebased
```

`timebased` の場合、期間は `statistics.time` で設定されます。

注意　これらの値は、統計をレポート・ファイルに出力する場合にのみ有効です。統計は、ロールオーバー時にのみサマリー・ファイルに出力されます。

statistics.time

統計がレポートされる間隔 (秒) を指定します。

次に例を示します。

```
binarywriter.statistics.time=5
```

statistics.tosummaryfile

各データ・ファイルに対する統計を含むサマリー・ファイルがロールオーバー時に作成されるかどうかを制御します。

次の例では、サマリー・ファイルが作成されます。

```
binarywriter.statistics.tosummaryfile=true
```

statistics.summary.fileformat

サマリー・ファイルの内容とその内容が書き込まれる順序を制御します。複数のカンマ区切りの ASCII 値が指定されます。

有効な値は次のとおりです。

schema: 統計の対象の表のスキーマまたは所有者

table: 統計の対象の表

schemaandtable: 1 列にピリオド '.' で区切ったスキーマと表

gtotal: ユーザー・イグジットの起動以降、指定された表について出力されたレコードの総数

gtotaldetail: デリミタで区切られた、ユーザー・イグジットの起動以降の挿入、更新および削除の総数

gctimestamp: 指定された表のユーザー・イグジットの起動以降の最小および最大コミット・タイムスタンプ

ctimestamp: 関連付けられたデータ・ファイル内の指定された表の最小および最大コミット・タイムスタンプ。

total: 関連付けられたデータ・ファイル内の指定された表のレコード出力の総数

totaldetail: 関連付けられたデータ・ファイル内の指定された表の挿入、更新および削除出力の総数

rate: 関連付けられたデータ・ファイル内の指定された表のデータ出力の平均速度 (レコード数 / 秒)

ratedetail: 関連付けられたデータ・ファイル内の指定された表の挿入、更新および削除の平均速度 (レコード数 / 秒)

次に例を示します。

```
binarywriter.statistics.summary.fileformat=
              schema,table,total,totaldetail,gctimestamp,ctimestamp
```

statistics.overall

追加の統計行がサマリー・ファイルに書き込まれるかどうかを制御します。行には、`statistics.summary.fileformat` プロパティを使用してユーザーによって定義された全体的な (すべての表にわたる) 統計が含まれます。

次の例では、この行が書き込まれます。

```
binarywriter.statistics.overall=true
```

statistics.summary.delimiter.chars/code、*statistics.summary.eol.chars/code*

サマリー・ファイルのフィールド・デリミタおよび行末デリミタのオーバーライド値を指定します。`chars` の場合、ASCII 値を使用し、`code` の場合、16 進値を使用します。デフォルトは、カンマ (,) デリミタと改行文字です。次に例を示します。

```
binarywriter.statistics.summary.delimiter.chars=|
binarywriter.statistics.summary.eol.code=0a0c
```

statistics.summary.extension

データ・ファイルごとに出力される統計サマリー・ファイルに使用される拡張子のオーバーライドを指定します。デフォルトは、`stats` です。

次の例では、拡張子が `.stats` から `.statistics` に変更されます。

```
binarywriter.statistics.summary.extension=.statistics
```

第 5 章

プロパティ・テンプレート

構成する各ファイル・ライターに設定可能なプロパティは多数あります。この作業を簡単に行うために、Oracle GoldenGate には特定のファイル・コンシューマ用の標準的なプロパティを含んだプロパティのテンプレートが用意されています。

プロパティ・テンプレートの使用

テンプレートを使用すると、共通の使用方法に基づいた特定のプロパティがあらかじめ設定されます。テンプレートの設定は、プロパティ・ファイルでのプロパティの設定によってオーバーライドされます。テンプレート内のプロパティごとに、システムは、まずそのプロパティがプロパティ・ファイル自体で設定されていないかチェックします。ユーザーが指定していない場合、テンプレートの設定が使用されます。

{writer}.template={template_name}

template プロパティを使用して、使用される標準的なプロパティのテンプレートの名前を指定します。構文は次のとおりです。

```
{writer}.template={template_name}
```

条件 : テンプレート名は、特定のファイル・コンシューマ用のデフォルト・プロパティ設定の名前付きファイルに対応します。有効なテンプレート名には、次のものなどがあります。

- SIEBEL: Siebel Remote によって消費されるトランザクション情報を含む DSV 形式の出力ファイルを作成するためのテンプレート。
- ABINITIO: Ab Initioによって消費されるLDV形式の出力を作成するためのテンプレート。
- GREENPLUM: GREENPLUMによって消費される、DSV形式の表ごとの出力ファイルを作成するためのテンプレート。
- NETEZZA: NETEZZAによって消費される、DSV形式の表ごとの出力ファイルを作成するためのテンプレート。
- COMMADELIM: カンマ区切りの表ごとの出力を作成するためのテンプレート。

デフォルト・プロパティ

すべてのライターは次のプロパティを使用します。各プロパティに示されている値はデフォルトです。

```
{writer}.files.data.rootdir=./out
{writer}.files.data.rollover.time=10
{writer}.files.data.rollover.size=100000
{writer}.files.data.norecords.timeout=10
{writer}.files.control.use=true
{writer}.files.control.ext=.ctrl
{writer}.files.control.rootdir=./out
```

Siebel Remote

```
goldengate.userexit.outputmode=txs
goldengate.userexit.buffertxs=true
goldengate.userexit.datetime.removecolon=true
goldengate.userexit.timestamp=utc
{writer}.mode=DSV
{writer}.rawchars=false
{writer}.includebefores=true
{writer}.includecolnames=true
{writer}.omitvalues=false
{writer}.diffsonly=false
{writer}.omitplaceholders=true
{writer}.files.onepertable=false
{writer}.files.data.ext=_data.csv
{writer}.files.data.tmpext=_data.csv.temp
{writer}.files.data.bom.code=efbbbf
{writer}.dsv.nullindicator.chars=NULL
{writer}.dsv.nullindicator.escaped.chars=
{writer}.dsv.fielddelim.chars=,
{writer}.dsv.fielddelim.escaped.chars=
{writer}.dsv.linedelim.chars=\n
{writer}.dsv.linedelim.escaped.chars=
{writer}.dsv.quotes.chars="
{writer}.dsv.quotes.escaped.chars=""
{writer}.dsv.quotealways=true
{writer}.groupcols=true
{writer}.afterfirst=true
{writer}.begintx.metacols="B","S",position,"GGMC",%LAST_UPD_BY,"1",
    numops
{writer}.metacols="R",opcode,%ROW_ID,%LAST_UPD_BY,%LAST_UPD,
%MODIFICATION_NUM,%CONFLICT_ID,position,txoppos,table,"","","","","",
    "",%DB_LAST_UPD,%DB_LAST_UPD_SRC,numcols
{writer}.metacols.DB_LAST_UPD.omit=true
{writer}.metacols.DB_LAST_UPD_SRC.omit=true
{writer}.metacols.opcode.updatepk.chars=U
{writer}.metacols.position.format=dec
{writer}.endtx.metacols="E"
```

Ab Initio

```
{writer}.mode=LDV
{writer}.files.onepertable=false
{writer}.files.data.ext=.data
{writer}.files.data.tmpext=.temp
{writer}.metacols=position,timestamp,opcode,txind,schema,table
{writer}.metacols.timestamp.fixedlen=26
{writer}.metacols.schema.fixedjustify=right
{writer}.metacols.schema.fixedpadchar.chars=Y
{writer}.metacols.opcode.fixedlen=1
{writer}.metacols.opcode.insert.chars=I
{writer}.metacols.opcode.update.chars=U
{writer}.metacols.opcode.delete.chars=D
{writer}.metacols.txind.fixedlen=1
{writer}.metacols.txind.begin.chars=B
{writer}.metacols.txind.middle.chars=M
{writer}.metacols.txind.end.chars=E
{writer}.metacols.txind.whole.chars=W
{writer}.metacols.position.format=dec
{writer}.ldv.vals.missing.chars=M
{writer}.ldv.vals.present.chars=P
{writer}.ldv.vals.null.chars=N
{writer}.ldv.lengths.record.mode=binary
{writer}.ldv.lengths.record.length=4
{writer}.ldv.lengths.field.mode=binary
{writer}.ldv.lengths.field.length=2
{writer}.statistics.period=onrollover
{writer}.statistics.tosummaryfile=true
{writer}.statistics.overall=true
{writer}.statistics.summary.fileformat=schema,table,schemaandtable,
    total,gctimestamp,ctimestamp
{writer}.statistics.summary.delimiter.chars=|
{writer}.statistics.summary.eol.chars=\n
```

Netezza

```
{writer}.mode=DSV
{writer}.rawchars=false
{writer}.includebefores=false
{writer}.includecolnames=false
{writer}.omitvalues=false
{writer}.diffsonly=false
{writer}.omitplaceholders=false
{writer}.files.onepertable=true
{writer}.files.data.ext=_data.dsv
{writer}.files.data.tmpext=_data.dsv.temp
{writer}.dsv.nullindicator.chars=
{writer}.dsv.fielddelim.chars=;
{writer}.dsv.fielddelim.escaped.chars=
```

Greenplum

```
{writer}.mode=DSV
{writer}.rawchars=false
{writer}.includebefores=false
{writer}.includecolnames=false
{writer}.omitvalues=false
{writer}.diffsonly=false
{writer}.omitplaceholders=false
{writer}.files.onepertable=true
{writer}.files.data.ext=_data.dsv
{writer}.files.data.tmpext=_data.dsv.temp
{writer}.dsv.nullindicator.chars=
{writer}.dsv.fielddelim.chars=|
{writer}.dsv.fielddelim.escaped.chars=
{writer}.metacols=opcode,timestamp
{writer}.metacols.opcode.insert.chars=I
{writer}.metacols.opcode.update.chars=U
{writer}.metacols.opcode.delete.chars=D
```

カンマ区切り

```
{writer}.mode=DSV
{writer}.rawchars=false
{writer}.includebefores=false
{writer}.includecolnames=false
{writer}.omitvalues=false
{writer}.diffsonly=false
{writer}.omitplaceholders=false
{writer}.files.onepertable=true
{writer}.files.data.ext=_data.dsv
{writer}.files.data.tmpext=_data.dsv.temp
{writer}.dsv.nullindicator.chars=NULL
{writer}.dsv.fielddelim.chars=,
{writer}.dsv.linedelim.chars=\n
{writer}.dsv.quotes.chars="
{writer}.dsv.quotes.escaped.chars=""
{writer}.metacols=position,txind,opcode,timestamp,schema,table
{writer}.statistics.period=onrollover
{writer}.statistics.overall=true
```

付録 1
アダプタ例

Oracle GoldenGate アダプタのインストールには、例が含まれています。次の例は、インストール場所の決まったサブディレクトリにあります。

FlatFileWriter

- Oracle GoldenGate フラット・ファイル・アダプタを使用して、Oracle GoldenGate 証跡データをテキスト・ファイルに変換します。

MessageDelivery

- Oracle GoldenGate Java アダプタを使用し、カスタム・メッセージ形式を使用して JMS メッセージを送信します。
- Oracle GoldenGate Java アダプタを使用し、カスタム・メッセージ・ヘッダー・プロパティを使用して JMS メッセージを送信します。

MessageCapture

- Oracle GoldenGate Java アダプタを使用して、JMS メッセージを処理し、Oracle GoldenGate 証跡を作成します。

JavaUserExitAPI

- Oracle GoldenGate Java アダプタ API を使用して、カスタム・イベント・ハンドラを記述します。

索引

セ

テ

ト

ナ

フ

メ

ラ

レ

ロ