Oracle® GoldenGate

Java アダプタ管理者ガイド

11*g* リリース 2 (11.2.1.0.0)

B71936-01 (原本部品番号 : E28384-01)

2012 年 12 月

ORACLE

Oracle GoldenGate Java アダプタ管理者ガイド 11g リリース 2 (11.2.1.0.0)

B71936-01 (原本部品番号 : E28384-01)

目次

第 1 章
概要

このガイドでは、次の点について説明します。

- Java 用 Oracle GoldenGate のインストール、構成および実行
- あらかじめビルドされた Java Message Service (JMS) およびファイル・ハンドラの使用
- カスタム・フィルタ、フォーマッタまたはイベント・ハンドラの開発

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。
- 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、プラットフォーム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 アダプタを Oracle GoldenGate コア製品と統合し、次のいずれかを行います。

- JMS メッセージを読み取り、Oracle GoldenGate 証跡として配信します。
- 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します。
- 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き込みます。

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あわ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にパブリッシュされます。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。

Oracle GoldenGate VAM メッセージの取得

Oracle GoldenGate メッセージ取得アダプタは JMS メッセージングに接続してメッセージを解析し、メッセージ・データから Oracle GoldenGate 証跡を作成する Oracle GoldenGate Extract に VAM インタフェースを介して送信します。これによって、ターゲット・データベース用に稼働している Oracle GoldenGate システムに JMS メッセージが配信されます。

Oracle GoldenGate JMS メッセージ取得を使用するには、次の 2 つのコンポーネントが必要です。

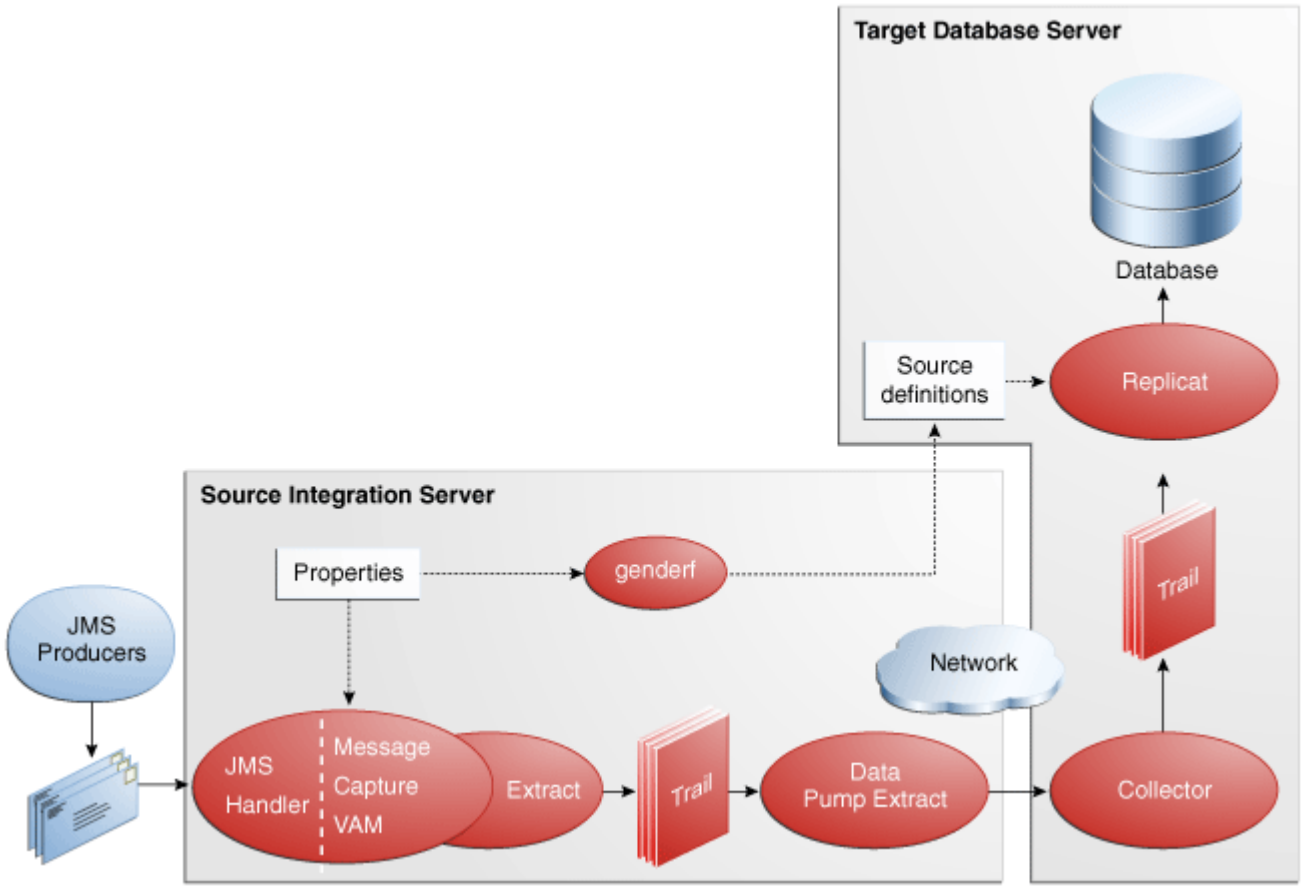
- 動的にリンクされた共有 VAM ライブラリ。Oracle GoldenGate Extract プロセスにアタッチされています。
- 個別のユーティリティである Gendef。メッセージ取得プロパティ・ファイルおよびパーサー固有のデータ定義を使用して Oracle GoldenGate ソース定義ファイルを作成します。

メッセージ取得構成オプション

メッセージ取得の 3 つの部分を構成するオプションは次のとおりです。

- メッセージ接続 : プロパティ・ファイルの値によって、JMS クライアントの Java クラスパス、JMS ソース宛先名、JNDI 接続プロパティ、セキュリティ情報などの接続プロパティが設定されます。
- 解析 : プロパティ・ファイルの値によって、固定幅、カンマ区切りまたは XML のメッセージの解析ルールが設定されます。これには、使用されるデリミタ、トランザクションの始まりと終わりの値、日時形式などの設定があります。
- VAM インタフェース : VAM .dll または .so ライブラリを特定するパラメータおよびプロパティ・ファイルが Oracle GoldenGate コア Extract プロセス用に設定されます。

図 1 JMS メッセージ取得用の構成



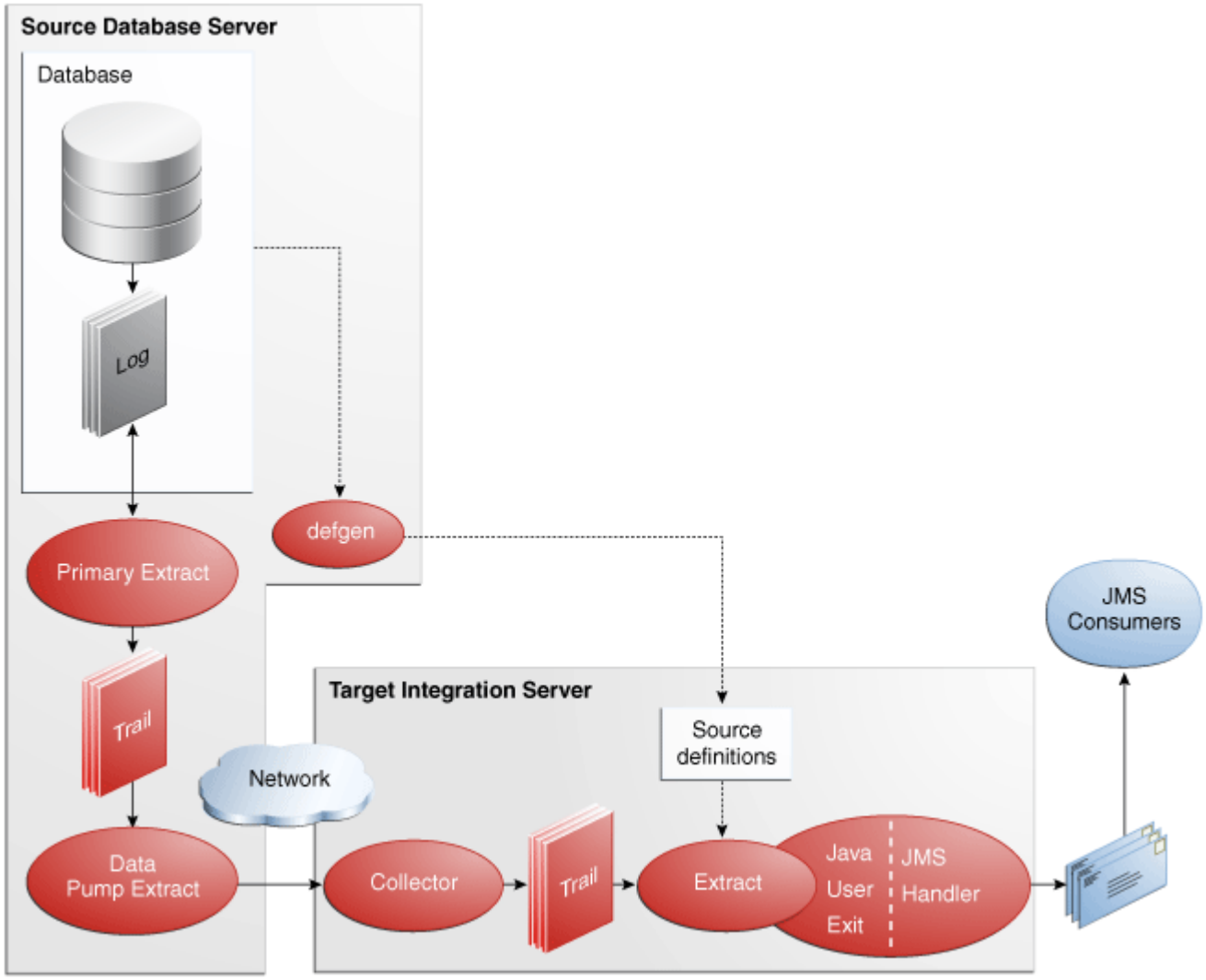
Oracle GoldenGate Java ユーザー・イグジット

Oracle GoldenGate によって取得されたトランザクション・データは、Oracle GoldenGate Java API を介してリレーショナル・データベース以外のターゲット (JMS (Java Message Service)、ディスクへのファイルの書込み、カスタム・アプリケーションの Java API との統合などの) に配信できます。

Java 用 Oracle GoldenGate には、Oracle GoldenGate Extract プロセスから Java で記述されたコードを実行する機能があります。Java 用 Oracle GoldenGate を使用するには、次の 2 つのコンポーネントが必要です。

- C/C++ で実装され、**ユーザー・イグジット** (UE) として C API を介して Oracle GoldenGate Extract プロセスと統合される、動的にリンクされたか、共有のライブラリ。
- Oracle GoldenGate Java API を構成する Java ライブラリのセット (jar)。この Java フレームワークは、Java Native Interface (JNI) を介してユーザー・イグジットと通信します。

図 2 JMS ハンドラを使用した構成



配信構成オプション

動的にリンクされたライブラリは、単純なプロパティ・ファイルを使用して構成できます。Java フレームワークは、このユーザー・イグジットによってロードされ、プロパティ・ファイルによって初期化もされます。アプリケーションの動作は、次のようにしてカスタマイズできます。

- プロパティ・ファイルの編集。たとえば、次のような編集があります。
  - ホスト名、ポート番号、出力ファイル名、JMS 接続設定の設定
  - トランザクションの送信先である任意の数のアクティブ・ハンドラをリストすることによるターゲット (JMS、ファイルなど) の追加 / 削除
  - デバッグレベルのロギングなどの有効化 / 無効化
  - 使用されるメッセージ形式の識別。

- JMS またはファイルに送信されるメッセージの形式のカスタマイズ。メッセージ形式は次のようにしてカスタマイズできます。
  - 既存のフォーマッタ (固定長またはフィールド区切りのメッセージ形式用) のプロパティの設定
  - Velocity テンプレート・マクロ言語を使用したメッセージ・テンプレートのカスタマイズ
  - (オプション) カスタム Java コードの記述。
- (オプション) トランザクションおよび操作のカスタム処理、フィルタリング、またはカスタム・メッセージ形式の実装を行うカスタム Java コードの記述。

JMS を介したメッセージの送信およびディスクへのファイルの書込み用に既存の実装 (ハンドラ) があります。メッセージの送信用にあらかじめ定義されたメッセージ形式がいくつかあります (XML、フィールド区切りなど)。あるいは、テンプレートを使用してカスタム形式を実装できます。各ハンドラには、構成プロパティを記述したドキュメントがあります。たとえば、ファイル・ライターに対するファイル名の指定や、JMS ハンドラに対する JMS キュー名の指定などがあります。一部のプロパティは複数のハンドラに適用されます。たとえば、同一のメッセージ形式を JMS とファイルに使用できます。

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

Java 用 Oracle GoldenGate のインストール

この章では、Oracle GoldenGate Java 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。

インストールの準備

適切な Java のバージョンがインストールされていることと、環境変数が適切に設定され、構成されていることを確認し、Java 環境を整えます。

Java のインストール

Java 用 Oracle GoldenGate のインストールと実行の前に、Java (JDK または JRE) バージョン 1.6 以上をインストールする必要があります。Java Runtime Environment (JRE) または完全 Java Development Kit (JRE が含まれる) を使用できます。

環境変数の設定

Java 用 Oracle GoldenGate 向けに Java 環境を構成するには、次のようにします。

- Java ランタイムを特定するための `PATH` 環境変数を構成します。
- 共有 (動的にリンクされた)Java 仮想マシン (JVM) ライブラリも特定される必要があります。

Windows ではこれらの環境変数はシステム変数として設定し、Linux/UNIX ではグローバルに、あるいは Oracle GoldenGate プロセスを実行するユーザーを対象に設定します。Windows および UNIX/Linux に対するこれらの環境変数の設定の例を次に示します。

注意 **Java のインストール時にインストールされる JVM には 2 つのバージョンがあります。1 つは `JAVA_HOME/.../client` で、もう 1 つは `JAVA_HOME/.../server` です。よりよいパフォーマンスを得るには、可能であればサーバー版を使用します。Windows では、JRE のみ (JDK ではなく) がインストールされた場合、クライアント JVM のみが存在する場合があります。**

Windows 上の Java

Java のインストール後、JRE および JVM DLL (`jvm.dll`) を特定するための `PATH` を構成します。

```
set JAVA_HOME=C:\Program Files\Java\jdk1.6.0
set PATH=%JAVA_HOME%\bin;%PATH%
set PATH=%JAVA_HOME%\jre\bin\server;%PATH%
```

前述の例では、ディレクトリ `%JAVA_HOME%\jre\bin\server` にファイル `jvm.dll` が含まれています。

コマンド・プロンプトを開いて Java のバージョンをチェックし、環境が設定されていることを確認します。

```
C:\> java -version
java version "1.6.0_30" Java(TM) SE Runtime Environment (build 1.6.0_30-b13)
```

Linux/UNIX 上の Java

システムに対して適切な環境変数を使用し、PATH の JRE および JVM 共有ライブラリを特定するための環境を構成します。たとえば、Linux(および Solaris など) では、次のように LD_LIBRARY_PATH を設定し、JVM 共有ライブラリを含むディレクトリを含めます (sh/ksh/bash 向け)。

```
export JAVA_HOME=/opt/jdk1.6
export PATH=${JAVA_HOME}/bin:${PATH}
export LD_LIBRARY_PATH=${JAVA_HOME}/jre/lib/i386/server:${LD_LIBRARY_PATH}
```

前述の例では、ディレクトリ $JAVA_HOME/jre/lib/i386/server にファイル libjvm.so が含まれています。JVM ライブラリを含む実際のディレクトリは、OS によって、あるいは 32 ビット JVM を使用するか、64 ビット JVM を使用するかによって異なります。

コマンド・プロンプトを開いて Java のバージョンをチェックし、環境が設定されていることを確認します。

```
$ java -version
java version "1.6.0_30"
Java(TM) SE Runtime Environment (build 1.6.0_30-b02)
```

Oracle GoldenGate Java 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に対応する Java 用 Oracle GoldenGate 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
- Oracle GoldenGate フラット・ファイル・アダプタ。このアダプタの詳細は、『*フラット・ファイル用 Oracle GoldenGate* Java 用管理者ガイド』を参照してください。
- 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}
   ```

   これによって、15 ページの表「インストール・ディレクトリ構造の例」に示すいくつかのサブディレクトリにファイルがダウンロードされます。

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
   ```

3. Manager パラメータ・ファイルを作成します。

   ```
   GGSCI> EDIT PARAM MGR
   ```

4. エディタを使用して Manager パラメータ・ファイルに行を追加し、Manager がリスニングするポートを指定します。次に例を示します。

   ```
   PORT 7801
   ```

5. GGSCI に移動し、Manager を起動して稼働していることを確認します。

   ```
   GGSCI>START MGR
   GGSCI>INFO MGR
   ```

6. Windows で Manager をサービスとして実行する場合、`jvm.dll` を含むようにシステム変数 `PATH` を設定し、Manager サービスを削除して再追加します。

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`
|-[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。

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
|-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 Java アダプタのアップグレード

Oracle GoldenGate Java アダプタのアップグレードには 2 種類あります。

- ソース・データベースから取得された変更を受信し、Oracle GoldenGate 証跡に書き込むアダプタ。
- JMS ソースから変更を受信するアダプタ

アップグレード手順は、それぞれ異なります。

ソース・データベースの取得

アダプタがソース・データベースから証跡データを受信する場合、次のアップグレード手順を使用します。

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、Velocity テンプレート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 jar ファイルまたはカスタム・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・ディレクトリで 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。

JMS 取得

アダプタが JMS ソースから変更を取得する場合、次の手順を使用してインストール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。
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 jar ファイルまたはカスタム・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. 新規インストール・ディレクトリに書き込むように新規インストール・ディレクトリの Extract 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, . . .
   ```

8. 以前のバージョンの古い Extract を停止します。

   ```
   GGSCI> STOP EXTRACT old_name
   ```

9. 新規 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. 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、アップグレードした JMS 取得プロセスによって生成される新規証跡から読み取るようダウンストリーム・プロセスを構成します。

第 3 章

VAM: メッセージ取得の構成

この章では、JMS メッセージを取得する VAM Extract の構成方法について説明します。

VAM Extract の構成

Java メッセージ取得アプリケーションを実行するには、次のものが必要です。

- Oracle GoldenGate Java アダプタ
- Extract プロセス
- メッセージ取得用に構成された Extract パラメータ・ファイル
- ソース定義ファイルなどの受信データ形式の説明。

Extract の追加

メッセージ取得 VAM を Oracle GoldenGate インストールに追加するには、GGSCI コマンドを使用して Extract と Extract が作成する証跡を追加します。

```
ADD EXTRACT jmsvam, VAM
ADD EXTTRAIL dirdat/id, EXTRACT jmsvam, MEGABYTES 100
```

プロセス名 (`jmsvam`) は、8 文字以下のプロセス名に置き換えます。証跡識別子 (`id`) は、任意の 2 文字です。

注意 `BEGIN`、`EXTRBA` などの Extract の位置を設定するコマンドは、メッセージ取得に対してサポートされません。Extract の再開では、常にメッセージ・キューの終わりからメッセージを読み取ります。

Extract パラメータの構成

Extract パラメータ・ファイルには、VAM の定義および起動に必要なパラメータが含まれます。VAM と通信するためのサンプル Extract パラメータを表に示します。

パラメータ	説明
`EXTRACT jmsvam`	Extract プロセスの名前。

パラメータ	説明
`VAM ggjava_vam.dll, PARAMS dirprm/jmsvam.properties`	VAM ライブラリの名前とプロパティ・ファイルの場所を指定します。VAM プロパティは、Oracle GoldenGate インストールの場所の `dirprm` ディレクトリにある必要があります。
`TRANLOGOPTIONS VAMCOMPATIBILITY 1`	VAM の元の実装が使用されること (1) を指定します。
`TRANLOGOPTIONS GETMETADATAFROMVAM`	メタデータが VAM によって送信されることを指定します。
`EXTTRAIL dirdat/id`	Extract が作成するターゲット証跡の識別子を指定します。
`TABLE OGG.*`	処理する表のリスト。表名にワイルドカードを使用できます。

メッセージ取得の構成

メッセージ取得は、VAM プロパティ・ファイルのプロパティによって構成されます。このファイルは、Extract `VAM` パラメータの `PARAMS` オプションによって識別され、ロギング特性、パーサーのマッピングおよび JMS 接続設定の決定に使用されます。

メッセージの接続と取得

JMS メッセージを処理するには、JMS インタフェースへの接続の構成、トランザクションのメッセージの取得と解析、各メッセージの証跡への書込み、トランザクションのコミットおよびメッセージのキューからの削除を行う必要があります。

JMS への接続

JMS への接続は、汎用 JMS インタフェースを介して行われます。プロパティを設定して次の接続の特性を構成できます。

- JMS クライアントの Java クラスパス
- JMS キューまたはトピック・ソース宛先の名前
- Java Naming and Directory Interface (JNDI) 接続プロパティ
  - 初期コンテキストの接続プロパティ
  - 接続ファクトリ名
  - 宛先名
- セキュリティ情報
  - JNDI 認証資格証明
  - JMS ユーザー名とパスワード

VAM と連携するよう構成された Extract プロセス (例の `jmsvam` など) は、起動時にメッセージ・システムに接続します。

注意 処理時に接続の問題があった場合に自動的に再起動されるよう、Extract を Manger の AUTORESTART リストに含めることができます。

現在、Oracle GoldenGate Java メッセージ取得アダプタでは JMS テキスト・メッセージのみサポートされます。

メッセージの取得

次のメッセージを要求されると、接続処理によって次の手順が実行されます。

- ローカル JMS トランザクションがまだ開始されていない場合は開始します。
- メッセージ・キューからメッセージを読み取ります。
- メッセージがないために読取りが失敗した場合、ファイル終端メッセージを返します。
- そうではない場合、メッセージのコンテンツを返します。

トランザクションの完了

トランザクションを構成するメッセージがすべて正常に取得および解析され、Oracle GoldenGate 証跡に書き込まれると、ローカル JMS トランザクションがコミットされ、メッセージがキューまたはトピックから削除されます。エラーが発生した場合、ローカル・トランザクションはロールバックされ、メッセージは JMS キューに残されます。

第 4 章

VAM: メッセージの解析

この章では、Oracle GoldenGate Java アダプタに含まれるパーサーのタイプと各パーサーの JMS テキスト・メッセージの翻訳方法について説明します。

解析の概要

パーサーの役割は、JMS テキスト・メッセージ・データとヘッダー・プロパティを適切なトランザクションと操作のセットに翻訳し、VAM インタフェースに渡すことです。これを行うために、パーサーは必ず特定のデータを検出する必要があります。

- トランザクション識別子
- シーケンス識別子
- タイムスタンプ
- 表名
- 操作タイプ
- 特定の表名と操作タイプに固有の列データ

構成で必要とされる場合、他のデータが使用されます。

- トランザクション・インジケータ
- トランザクション名
- トランザクションの所有者

パーサーは、このデータを JMS ヘッダー・プロパティ、システム生成値、静的な値から、またはパーサー固有の方法で取得できます。これは、情報の性質によって異なります。

パーサー・タイプ

Oracle GoldenGate メッセージ取得アダプタでは、3 種類のパーサーがサポートされます。

- 固定 : メッセージは、連続したテキストに固定幅フィールドとして表されるデータを含みます。
- 区切り : メッセージは、フィールドとレコード終端文字で区切られたデータを含みます。
- XML: メッセージは、XPath 式でアクセスされる XML データを含みます。

ソースおよびターゲットのデータ定義

プロパティと外部データを組み合せてソース・データ定義を定義する方法はいくつかあります。Oracle GoldenGate Gendef ユーティリティでは、これらのデータ定義とパーサー・プロパティに基づいて標準ソース定義ファイルを生成します。オプションはパーサー・タイプによって異なります。

- 固定 : COBOL コピーブック、ソース定義またはユーザー定義
- 区切り : ソース定義またはユーザー定義
- XML: ソース定義またはユーザー定義

選択されたパーサーのデータの取得方法およびソース定義のターゲット定義への変換方法を構成するプロパティがいくつかあります。

必須データ

パーサーがメッセージを翻訳するには、次の情報が必要です。

トランザクション識別子

トランザクション識別子 (`txid`) によって、Oracle GoldenGate 証跡ファイルに書き込まれる操作はトランザクションにグループ化されます。Oracle GoldenGate メッセージ取得アダプタでは、連続し、インターリーブされていないトランザクションのみサポートされます。トランザクション識別子は、トランザクションごとに増分される一意の値です。通常、システム生成値が使用されます。

シーケンス識別子

シーケンス識別子 (`seqid`) によって、各操作が内部で識別されます。これは、リカバリ処理時に使用され、Oracle GoldenGate 証跡に書込み済の操作が識別されます。シーケンス識別子は、操作ごとに増分される一意の値です。長さは固定です。

プロバイダのメッセージ識別子が増分され、一意の場合、JMS メッセージ ID をシーケンス識別子として使用できます。ただし、JMS でメッセージの順序が保証されない場合 (クラスタリングの使用や失敗したトランザクションなど) や ID は一意ではあるが、増分されない場合があります。システム生成のシーケンス ID を使用できますが、リカバリの状況によってはメッセージが重複する場合があります。推奨される方法は、メッセージをキューに追加する JMS クライアントでメッセージ ID、ヘッダー・プロパティまたはデータ要素をアプリケーションで生成される一意の増分値に設定することです。

タイムスタンプ

タイムスタンプ (`timestamp`) は、Oracle GoldenGate 証跡内で操作のコミット・タイムスタンプとして使用されます。増分されますが、増分が必須ではありません。トランザクションまたは操作間で一意である必要はありません。解析可能な任意のデータ形式でかまいません。

表名

表名を使用して、列データが属する論理表が識別されます。アダプタは、`SCHEMA_NAME.TABLE_NAME` 形式の 2 つの部分で構成される表名を必要とします。これは、個別に (`schema` **および** `table`) 定義するか、スキーマと表の組合せとして (`schemaandtable`) 定義します。

1 つのフィールドにスキーマ名と表名の両方が含まれることも、スキーマ名と表名が別々に含まれることも、スキーマがソフトウェア・コードに含まれ、表名のみが必要なこともあります。スキーマ名と表名の指定方法は、パーサーによって異なります。いずれの場合も、2 つの部分で構成される論理表名を使用して Oracle GoldenGate 証跡にレコードが書き込まれ、証跡を表すソース定義ファイルが生成されます。

操作タイプ

操作タイプ (optype) を使用して、Oracle GoldenGate 証跡への書込み時に操作が挿入、更新、削除のいずれかが決定されます。特定の操作の操作タイプ値は、各操作タイプに定義された値に一致します。

各操作タイプについて Oracle GoldenGate 証跡に書き込まれるデータは、Extract の構成によって異なります。

- 挿入
  - すべての列のアフター値が証跡に書き込まれます。
- 更新
  - デフォルト : キーのアフター値が書き込まれます。ビフォア値が存在し、比較可能な場合、変更された列のアフター値が書き込まれます。ビフォア値がない場合、すべての列が書き込まれます。
  - NOCOMPRESSUPDATES: すべての列のアフター値が証跡に書き込まれます。
  - GETUPDATEBEFORES: ビフォア値が存在し、比較可能な場合、変更された列のビフォア値とアフター値が証跡に書き込まれます。ビフォア値がない場合、アフター値のみが書き込まれます。
  - NOCOMPRESSUPDATES と GETUPDATEBEFORES の両方が含まれ、ビフォア値が存在する場合、すべての列のビフォア値とアフター値が証跡に書き込まれます。
- 削除
  - デフォルト : すべてのキーのビフォア値が証跡に書き込まれます。
  - NOCOMPRESSDELETES: すべての列のビフォア値が証跡に書き込まれます。

キーのビフォア値が存在し、アフター値と一致しない場合、主キー更新操作も生成されます。

列データ

すべてのパーサーは列データをメッセージ・テキストから取得し、Oracle GoldenGate 証跡に書き込みます。列は、ソース定義で定義されている索引順に読み取られる場合もあれば、名前でアクセスされる場合もあります。

構成および元のメッセージ・テキストに応じて、列データのビフォア・イメージとアフター・イメージの両方が使用可能な場合とアフター・イメージのみが使用可能な場合があります。更新の場合、更新されていない列が使用可能な場合と使用可能でない場合があります。

すべての列データはテキストとして取得されます。ソース定義に基づいて、その列の正しいデータ型に内部で変換されます。変換に問題があるとエラーになり、プロセスは異常終了します。

オプション・データ

次のデータを含めることができますが、必須ではありません。

トランザクション・インジケータ

トランザクションとメッセージの関係には次のものがあります。

- 各メッセージに 1 つのトランザクション
  これは、メッセージのスコープによって自動的に決定されます。

- 各メッセージに複数のトランザクション

  これは、トランザクション・インジケータ (txind) によって決定されます。トランザクション・インジケータがない場合、XML パーサーは、適合するトランザクション・ルールに基づいてトランザクションを作成できます。

- 各トランザクションに複数のメッセージ

  操作がトランザクション全体の始まりか、中間か、終わりかを指定するために、トランザクション・インジケータ (txind) が必要です。特定の操作のトランザクション・インジケータは、各トランザクション・インジケータ・タイプに定義された値に一致します。インジケータ値が始まりまたは全体の場合はトランザクションが開始され、中間の場合は続行されます。終わりまたは全体の場合は終了されます。

トランザクション名

トランザクション名 (txname) は、任意の名前とトランザクションとの関連付けに使用できるオプション・データです。これは、GETENV 関数を使用してトークンとして証跡に追加できます。

トランザクションの所有者

トランザクションの所有者 (txowner) は、任意のユーザー名とトランザクションとの関連付けに使用できるオプション・データです。これは、GETENV 関数を使用してトークンとして証跡に追加したり、EXCLUDEUSER Extract パラメータを使用して特定のトランザクションを処理から除外するために使用できます。

固定幅解析

固定幅解析は、各フィールドの位置と長さを定義するデータ定義に基づきます。これは、COBOL コピーブックの形式です。プロパティのセットで、コピーブックを Oracle GoldenGate 証跡およびソース定義ファイルの論理レコードにマップするためのルールを定義します。

受信データは、標準形式のヘッダーとその後に続くデータ・セグメントで構成されます。両方とも固定幅のフィールドを含みます。データは、コピーブックの PIC 定義に基づいて解析されます。27 ページの「ヘッダーとレコード・データ型の翻訳」の説明のように翻訳されて証跡に書き込まれます。

ヘッダー

ヘッダーは、次の情報を含むコピーブック 01 レベル・レコードによって定義される必要があります。

- レコードのコミット・タイムスタンプまたは変更時間
- 操作のタイプ (挿入、更新または削除) を示すコード
- データ・セグメントの解析時使用するコピーブック・レコード名

Oracle GoldenGate ヘッダー・フィールドにマップされないヘッダー・レコードのフィールドは列として出力されます。

ヘッダーの指定

次の例では、必須ヘッダー値を含むコピーブック定義を示します。

```
01 HEADER.
    20 Hdr-Timestamp          PIC X(23)
    20 Hdr-Source-DB-Function PIC X
    20 Hdr-Source-DB-Rec-ID   PIC X(8)
```

この例に対して次のプロパティを設定します。

```
fixed.header=HEADER
fixed.timestamp=Hdr-Timestamp
fixed.optype=Hdr-Source-DB-Function
fixed.table=Hdr-Source-DB-Rec-Id
```

この場合の論理名表出力は、`Hdr-Source-DB-Rec-Id` という値になります。

複合表名の指定

複数のフィールドを表名に使用できます。たとえば、次のような静的プロパティを介して論理スキーマ名を定義できます。

```
fixed.schema=MYSCHEMA
```

コピーブック・ヘッダー定義から複数フィールドのデータ・レコードを定義するプロパティを追加できます。

```
01  HEADER.
    20  Hdr-Source-DB              PIC X(8).
    20  Hdr-Source-DB-Rec-Id       PIC X(8).
    20  Hdr-Source-DB-Rec-Version  PIC 9(4).
    20  Hdr-Source-DB-Function     PIC X.
    20  Hdr-Timestamp              PIC X(22).

fixed.header=HEADER
fixed.table=Hdr-Source-DB-Rec-Id,Hdr-Source-DB-Rec-Version
fixed.schema=MYSCHEMA
```

フィールドが連結されて、次のような論理スキーマと表名になります。

```
MYSCHEMA.Hdr-Source-DB-Rec-Id+Hdr-Source-DB-Rec-Version
```

タイムスタンプ形式の指定

タイムスタンプは、デフォルトの形式 `YYYY-MM-DD HH:MM:SS.FFF` (`FFF` はフィールドのサイズよって異なる) を使用して解析されます。

次の例に示すように日時フィールドの前にコメントを入力し、異なる受信形式を指定します。

```
01  HEADER.
* DATEFORMAT YYYY-MM-DD-HH.MM.SS.FF
    20  Hdr-Timestamp      PIC X(23)
```

関数の指定

プロパティを使用して、標準の Oracle GoldenGate 操作タイプを optype 値にマップします。次の例では、操作タイプは Hdr-Source-DB-Function フィールドにあることと、挿入の値は A、更新は U、削除は D であることを指定します。

```
fixed.optype=Hdr-Source-DB-Function
fixed.optype.insert=A
fixed.optype.update=U
fixed.optype.delete=D
```

ヘッダーとレコード・データ型の翻訳

ヘッダーのデータとレコード・データは、翻訳されたデータ型に基づいて証跡に書き込まれます。

- 日付形式のコメントが前に付いたフィールド定義は、指定されたサイズの Oracle GoldenGate 日時フィールドに翻訳されます。日付形式のコメントがない場合、フィールドは基になるデータ型によって定義されます。
- PIC X フィールドは、指定されたサイズの CHAR データ型に翻訳されます。
- PIC 9 フィールドは、定義された精度と位取りの NUMBER データ型に翻訳されます。符号付きの数値、符号なしの数値、小数部のある数値、小数部のない数値がサポートされます。

次の例では、様々な PIC 定義に対する翻訳を示します。

入力	出力
PIC XX	CHAR(2)
PIC X(16)	CHAR(16)
PIC 9(4)	NUMBER(4)
* YYMMDD PIC 9(6)	DATE(10) YYYY-MM-DD
PIC 99.99	NUMBER(4,2)
PIC 9(5)V99	NUMBER(7,2)

この例で、入力 YYMMDD 日付の 100522 は、2010-05-22 に翻訳されます。指定された PIC 9(5)V99 形式の数値 1234567 は、小数部が 2 桁の 7 桁の数値 12345.67 に翻訳されます。

キー識別子

コメントを使用して、データ・レコード内のキー列を識別します。ソース定義を生成する Gendef ユーティリティは、コメントを使用してキー列を特定します。

次の例では、Account が TABLE1 のキー列としてマークされています。

```
01 TABLE1
* KEY
20  Account     PIC X(19)
20  PAN_Seq_Num PIC 9(3)
```

区切り解析

区切り解析は、既存のソース定義ファイルとプロパティのセットに基づきます。プロパティで、使用するデリミタと列名とビフォア値があるかどうかなどの他のルールが指定されます。ソース定義ファイルによって、処理される有効な表、および表内の列の順序とデータ型が決まります。

区切りメッセージの形式は次のとおりです。

```
{METACOLS}n[,{COLNAMES}]m[,{COLBEFOREVALS}]m,{COLVALUES}m\n
```

条件: メタデータ列は *n* 個含めることができ、各メタデータ列の後に形式文に示すカンマなどのフィールド・デリミタが続きます。

列値は *m* 個含めることができます。各々、カンマなどのフィールド・デリミタがその前に付きます。

列名とビフォア値はオプションです。

各レコードは、`\n` などの行末デリミタで終わります。

メタデータ列

メタデータ列はヘッダーに相当し、特別な意味を持つフィールドを含みます。メタデータ列には、次の情報が含まれます。

- **optype** には、レコードが挿入か、更新か、削除かを示す値が含まれます。デフォルト値は、`I`、`U` および `D` です。
- **timestamp** は、レコードのコミット・タイムスタンプに使用する値の型を示します。タイムスタンプの形式のデフォルトは、`YYYY-DD-MM HH:MM:SS.FFF` です。
- **schemaandtable** は、`SCHEMA.TABLE` 形式のレコードの完全表名です。
- **schema** は、レコードのスキーマ名です。
- **table** は、レコードの表名です。
- **txind** は、レコードがトランザクションの始まりか、中間か、終わりか、唯一のレコードかを示す値です。デフォルト値は、0、1、2、3 です。
- **id** は、レコードのシーケンス番号 (RSN または CSN) として使用される値です。トランザクションの最初のレコード (操作) の id は、トランザクションのシーケンス番号に使用されます。

解析プロパティ

デリミタ、値および日時の形式を表すプロパティを設定できます。

デリミタを表すプロパティ

次のプロパティは、レコードを区切るための解析ルールを決定します。

- **fielddelim** は、1 つ以上の ASCII または 16 進文字をフィールド・デリミタの値として指定します。
- **recorddelim** は、1 つ以上の ASCII または 16 進文字をレコード・デリミタの値として指定します。
- **quote** は、引用符付きの値に使用する 1 つ以上の ASCII または 16 進文字を指定します。
- **nullindicator** は、NULL 値に使用する 1 つ以上の ASCII または 16 進文字を指定します。

デリミタのエスケープ文字を定義し、その文字がテキスト内にあった場合に置き換えられるようにすることができます。たとえば、バックスラッシュとアポストロフィ (\') が指定されている場合、入力 "They used Mike\'s truck" は "They used Mike's truck" に翻訳されます。また、2 つの引用符 ("") が指定されている場合、"They call him ""Big Al""" は "They call him "Big Al"" に翻訳されます。

レコード内に引用符なしのデータ値が存在することがありますが、引用符付きの値内のエスケープ文字のみシステムで削除されます。null インジケータと一致する引用符なしの文字列は、null として処理されます。

値を表すプロパティ

次のプロパティで詳細情報が提供されます。

- **hasbefores** は、各レコードにビフォア値が存在することを示します。
- **hasnames** は、各レコードに列名が存在することを示します。
- **afterfirst** は、列のアフター値がビフォア値の前に来ることを示します。
- **isgrouped** は、すべての列名、ビフォア値およびアフター値が列ごとに順にではなく、3 つのブロックにまとめられることを示します。

日時を表すプロパティ

デフォルトの形式 `YYYY-DD-MM HH:MM:SS.FFF` がデータの解析に使用されます。ユーザーはプロパティを使用してグローバル、表または列レベルでこれをオーバーライドできます。形式の変更の例を次に示します。

```
delim.dateformat.default=MM/DD/YYYY-HH:MM:SS
delim.dateformat.MY.TABLE=DD/MMM/YYYY
delim.dateformat.MY.TABLE.COL1=MMYYYY
```

解析手順

区切り解析の手順は次のとおりです。

1. パーサーはまず各レコードのメタデータ列を読み取り、確認します。
2. これによって表名が取得され、これを使用してソース定義ファイル内の表の列定義が検索されます。
3. 表の定義が見つからない場合、処理は停止します。
4. それ以外の場合、列が解析され、ソース定義に定義されている順序と形式で証跡に出力されます。

XML 解析

XML 解析は、既存のソース定義ファイルとプロパティのセットに基づきます。プロパティによって、トランザクション、操作および列に対応する XML 要素および属性を決定するルールが指定されます。ソース定義ファイルによって、処理される有効な表、およびそれらの表内の列の順序とデータ型が決まります。

XML のスタイル

XML メッセージは動的または静的な XML にフォーマットできます。実行時、*動的 XML* のコンテンツは、サンプル XML または XSD ドキュメントを使用してあらかじめ決定できないデータ値です。表および列の要素または属性名を決定する*静的 XML* のコンテンツは、それらのサンプル・ドキュメントを使用してあらかじめ決定できます。

次の 2 つの例には同じデータが含まれています。

静的XML のサンプル

```
<NewMyTableEntries>
  <NewMyTableEntry>
    <CreateTime>2010-02-05:10:11:21</CreateTime>
    <KeyCol>keyval</KeyCol>
    <Col1>col1val</Col1>
  </NewMyTableEntry>
</NewMyTableEntries>
```

NewMyTableEntries 要素は、トランザクション境界をマークします。NewMyTableEntry は、MY.TABLE への挿入を表します。タイムスタンプは要素テキスト値内にあり、列名は要素名で表されます。

プロパティ・ファイルにこれらの 2 つのスタイルの XML を、XPath ライクなプロパティを介して解析するルールを定義できます。プロパティの目的は、XPath 一致を介してあらかじめ定義されたソース定義ファイルに XML をマップすることです。

動的XML のサンプル

```
<transaction id="1234" ts="2010-02-05:10:11:21">
  <operation table="MY.TABLE" optype="I">
    <column name="keycol" index="0">
      <aftervalue><![CDATA[keyval]]></aftervalue>
    </column>
    <column name="col1" index="1">
      <aftervalue><![CDATA[col1val]]></aftervalue>
    </column>
  </operation>
</transaction>
```

各表に対する各操作は同じ構造を持ち、トランザクション、操作および列の各要素で構成される基本的なメッセージ構造です。表名、操作タイプ、タイムスタンプ、列名、列値などは属性または要素テキスト値から取得されます。

XML 解析ルール

XML のスタイルに関係なく、解析プロセスは次の事項を決定する必要があります。

- トランザクション境界
- 次のような操作エントリとメタデータ
  - 表名
  - 操作タイプ
  - タイムスタンプ
- 次のような操作エントリとメタデータ
  - 列名または索引のいずれか。両方が指定されている場合、システムは、指定されたデータを含む列の名前が指定された名前かをチェックします。
  - 列のビフォア値またはアフター値 (両方の場合も)。

これは、相互に関係のあるルールのセットを介して行われます。処理される各タイプの XML メッセージに対して、必要なデータの取得に使用されるルールを指定します。指定されたこれらの各ルールに対し、次の目的のプロパティを追加します。

- トランザクション、操作または列ルール・タイプとしてルールを指定します。どのタイプのルールも、指定された名前とタイプが必要です。
- 処理されるドキュメントに対してルールがアクティブかどうかを確認するために一致させるXPath式を指定します。これはオプションです。定義されていない場合、パーサーは親ルールのノードを一致させるか、これが最初のルールの場合はドキュメント全体のノードを一致させます。
- リストされた順で処理される詳細ルール (サブルール) をリストします。有効なサブルールは、ルール・タイプによって決まります。サブルールはオプションです。

次の例では、最上位レベルのルールは `genericrule` と定義されています。これは、`transaction` タイプ・ルールです。そのサブルールは `oprule` で定義され、このタイプは `operation` です。

```
xmlparser.rules=genericrule
xmlparser.rules.genericrule.type=tx
xmlparser.rules.genericrule.subrules=oprule
xmlparser.rules.oprule.type=op
```

XPath 式

XML パーサーでは、要素の一致およびデータの抽出に必要な XPath 式のサブセットがサポートされます。式は、特定の要素の一致またはデータの抽出に使用されます。

データ抽出の際、パスの大部分が一致に使用されます。式の末尾が抽出に使用されます。

サポートされる構文:

`/e`	ドキュメントのルートからの絶対パスを使用して `e` を一致させます。
`./e` または `e`	処理対象の現在のノードからの相対パスを使用して `e` を一致させます。
`../e`	現在のノードの親に基づいたパスを使用して (繰返し可能)`e` を一致させます。
`//e`	`e` がドキュメント内にあれば一致させます。
`*`	任意の要素を一致させます。注意 : 部分的にワイルドカードが使用された名前はサポートされません。
`[n]`	式の n 番目のオカレンスを一致させます。
`[x=v]`	x が値 v に等しい場合、一致させます。ここで、x は次のいずれかです。 ◆ `@att`: 属性値 ◆ `text()`: テキスト値 ◆ `name()`: 要素名 ◆ `position()`: 要素の位置

サポートされる式

ルート要素の一致	`/My/Element`
サブ要素と現在のノードの一致	`./Sub/Element`

n 番目の要素の一致	`/My/*[n]`
n 番目の Some 要素の一致	`/My/Some[n]`
任意のテキスト値の一致	`/My/*[text() ='value']`
Some 要素のテキストの一致	`/My/Some[text() = 'value']`
任意の属性の一致	`/My/*[@att = 'value']`
Some 要素の属性の一致	`/My/Some[@att = 'value']`

データ値の取得

パスの一致以外に、XPath 式は、データ値の取得にも使用できます (絶対パス、処理対象の現在のノードとの相対パスのいずれも)。データ値の式には前述のパス式を含めることができますが、後述のいずれかの値アクセッサで終わる必要があります。

`@att`	属性値。
`text()`	要素のテキスト・コンテンツ (値)。
`content()`	任意の子 XML ノードを含めた、要素のフル・コンテンツ。
`name()`	要素の名前。
`position()`	親内の要素の位置。

例を示します。

相対要素テキスト値を抽出する場合 :

```
/My/Element/text()
```

絶対属性値を抽出する場合 :

```
/My/Element/@att
```

一致を使用して要素テキスト値を抽出する場合 :

```
/My/Some[@att = 'value']/Sub/text()
```

注意　ancestor、descendent、self などのパス・アクセッサはサポートされません。

他の値式

XML パーサーによって抽出される値は、列値、またはトランザクションまたは操作のプロパティ (表、タイムスタンプなど) です。これらの値は XPath を使用するか、JMS メッセージのプロパティ、システム値またはハードコードされた値を介して XML から取得されます。XML パーサー・プロパティは、これらのオプションのうち、そのプロパティの値を取得するために有効なオプションを指定します。

次の例では、`timestamp` は、XPath 式、JMS プロパティまたはシステム生成タイムスタンプであることを指定します。

```
{txrule}.timestamp={xpath-expression}|${jms-property}|*ts
```

次の例では、table は、XPath 式、JMS プロパティまたはハードコードされた値であることを指定します。

```
{oprule}.table={xpath-expression}|${jms-property}|"value"
```

次の例では、name は、XPath 式またはハードコードされた値であることを指定します。

```
{colrule}.timestamp={xpath-expression}|"value"
```

トランザクション・ルール

トランザクションの境界を指定するルールが最上位のものです。メッセージには、1 つのトランザクション、複数のトランザクションまたは複数メッセージにわたるトランザクションの一部を含めることができます。これらは次のように指定されます。

- **single**: トランザクション・ルール一致は定義されません。
- **multiple**: 各トランザクション・ルール一致は新規トランザクションを定義します。
- **span**: トランザクション・ルールは定義されません。かわりに、トランザクション・インジケータが操作ルールに指定されます。

トランザクション・ルールの場合、XPath または他の式を介して次のルールのプロパティも定義できます。

- **timestamp**: トランザクションが発生した時間。
- **txid**: トランザクションの識別子。

トランザクション・ルールは複数のサブルールを持つことができますが、各サブルールは操作タイプのルールである必要があります。

例

次の例では、メッセージ全体であるトランザクションを指定し、JMS プロパティから取得されるタイムスタンプを含めます。

```
singletxrule.timestamp=$JMSTimeStamp
```

次の例はルート要素トランザクションを一致させ、ts 属性からタイムスタンプを取得します。

```
dyntxrule.match=/Transaction
dyntxrule.timestamp=@ts
```

操作ルール

操作ルールはトランザクション・ルールのサブルールまたは最上位ルール (トランザクションが操作のプロパティの場合) のいずれかです。

標準ルール・プロパティ以外に、操作ルールは、XPath または他の式を介して次のものも定義します。

- **timestamp**: 操作のタイムスタンプ。トランザクション・ルールが定義されている場合、これはオプションです。
- **table**: この操作の対象の表の名前。これをスキーマと組み合せて使用します。
- **schema**: 表のスキーマの名前。
- **schemaandtable**: SCHEMA.TABLE の形式でスキーマ名と表名の両方。これは、個々の表およびスキーマのプロパティのかわりに使用できます。

- **optype**: optype 値に基づいて、これが挿入、更新、削除のいずれの操作かを指定します。
  - **optype.insertval**: 挿入を表す値。デフォルトは、`I` です。
  - **optype.updateval**: 更新を表す値。デフォルトは、`U` です。
  - **optype.deleteval**: 削除を表す値。デフォルトは、`D` です。
- **seqid**: 操作の識別子。`txid` がトランザクション・レベルで定義されていない場合、これはトランザクション識別子になります。
- **txind**: この操作がトランザクションの始まりか、中間か、終わりか、操作全体かを指定します。このプロパティはオプションで、操作ルールがトランザクション・ルールのサブルール場合、無効です。

操作ルールは、操作タイプまたは列タイプの複数のサブルールを持つことができます。

例

次の例では、`/Transaction` の `/Operation` 要素から操作情報を動的に取得します。

```
dynoprule.match=./Operation
dynoprule.schemaandtable=@table
dynoprule.optype=@type
```

次の例は、`/NewMyTableEntry` 要素を `MY.TABLE` 表に対する挿入操作に静的に一致させます。

```
statoprule.match=./NewMyTableEntry
statoprule.schemaandtable="MY.TABLE"
statoprule.optype="I"
statoprule.timestamp=./CreateTime/text()
```

列ルール

列ルールは、操作ルールのサブルールである必要があります。標準ルール・プロパティ以外に、列ルールは、XPath または他の式を介して次のものも定義します。

- **name**: 表定義内の列の名前。
- **index**: 表定義内の列の索引。

> **注意** `name`、`index` **のいずれか一方のみが定義されている場合、他方が決定されます。**

- **before.value**: 列のビフォア値。これは、削除の場合必須ですが、更新の場合オプションです。
- **before.isnull**: 列のビフォア値が null かどうかを示します。
- **before.ismissing**: 列のビフォア値が欠落しているかどうかを示します。
- **after.value**: 列のビフォア値。これは、削除の場合必須ですが、更新の場合オプションです。
- **after.isnull**: 列のビフォア値が null かどうかを示します。
- **after.ismissing**: 列のビフォア値が欠落しているかどうかを示します。
- **value**: 特定のビフォア値またはアフター値でオーバーライドされない場合に `before.value` と `after.value` の両方に使用する式。これは、更新に対する異なるビフォア値をサポートしないことに注意してください。

- **isnull**: オーバーライドされない場合に before.isnull と after.isnull の両方に使用する式。
- **ismissing**: オーバーライドされない場合に before.ismissing と after.ismissing の両方に使用する式。

例

次の例では、/Operation の /Column 要素から列情報を動的に取得します。

```
dyncolrule.match=./Column
dyncolrule.name=@name
dyncolrule.before.value=./beforevalue/text()
dyncolrule.after.value=./aftervalue/text()
```

次の例は、/KeyCol および /Col1 要素を MY.TABLE の列に静的に一致させます。

```
statkeycolrule.match=/KeyCol
statkeycolrule.name="keycol"
statkeycolrule.value=./text()
statcol1rule.match=/Col1
statcol1rule.name="col1"
statcol1rule.value=./text()
```

ルール全体の例

次の例では、前述の XML サンプルおよび適切なルールを使用して、MY.TABLE に対する同じ結果の操作を生成します。

動的 XML

```
<transaction id="1234"
    ts="2010-02-05:10:11:21">
  <operation table="MY.TABLE" optype="I">
    <column name="keycol" index="0">
      <aftervalue>
<![CDATA[keyval]]>
      </aftervalue>
    </column>
    <column name="col1" index="1">
      <aftervalue>
         <![CDATA[col1val]]>
      </aftervalue>
    </column>
  </operation>
</transaction>
```

静的 XML

```
NewMyTableEntries>
  <NewMyTableEntry>
    <CreateTime>
       2010-02-05:10:11:21
    </CreateTime>
    <KeyCol>keyval</KeyCol>
    <Col1>col1val</Col1>
  </NewMyTableEntry>
</NewMyTableEntries>
```

```
dyntxrule.match=/Transaction
dyntxrule.timestamp=@ts
dyntxrule.subrules=dynoprule
dynoprule.match=./Operation
dynoprule.schemaandtable=@table
dynoprule.optype=@type
dynoprule.subrules=dyncolrule
dyncolrule.match=./Column
dyncolrule.name=@name
```

```
stattxrule.match=/NewMyTableEntries
stattxrule.subrules= statoprule
statoprule.match=./NewMyTableEntry
statoprule.schemaandtable="MY.TABLE"
statoprule.optype="I"
statoprule.timestamp=./CreateTime/text()
statoprule.subrules= statkeycolrule,
statcol1rule
statkeycolrule.match=/KeyCol
```

```
dyncolrule.before.value=./beforevalue/text()
dyncolrule.after.value=./aftervalue/text()
```

```
statkeycolrule.name="keycol"
statkeycolrule.value=./text()
statcol1rule.match=/Col1
statcol1rule.name="col1"
statcol1rule.value=./text()
```

```
INSERT INTO MY.TABLE (KEYCOL, COL1)
VALUES ('keyval', 'col1val')
```

ソース定義生成ユーティリティ

Java 用 Oracle GoldenGate には、プロパティ・ファイルに定義されているプロパティから Oracle GoldenGate ソース定義ファイルを生成する Gendef ユーティリティが含まれています。プロパティ設定および他のパーサー固有のデータ定義値に基づいて、表の標準化された定義を作成します。

このユーティリティを実行する構文は次のとおりです。

```
gendef prop {property_file} [-out {output_file}]
```

これは、デフォルトではソース定義を標準出力に送信しますが、out パラメータを使用してファイルに仕向けることができます。次に例を示します。

```
gendef prop dirprm/jmsvam.properties -out dirdef/msgdefs.def
```

出力ソース定義ファイルをポンプまたは配信プロセスで使用し、VAM を介して作成された証跡データを解釈します。

第 5 章

VAM: メッセージ取得プロパティ

この章では、Java 用 Oracle GoldenGate VAM のプロパティ・ファイルの構成に使用できるオプションについて説明します。

このプロパティ・ファイルは、Oracle GoldenGate インストールの場所の dirprm ディレクトリに配置します。VAM プロパティ・ファイルの名前は、Extract VAM パラメータで設定されます。

プロパティ・ファイル内のプロパティにはすべて fully.qualified.name=value の形式が使用されます。値は、整数、ブール値、1 つの文字列またはカンマ区切りの文字列です。

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・アウトできます。ただし、行末にコメントを置くことはできません。行全体をコメントにするか、プロパティをコメントにします。

ロギングおよび接続のプロパティ

次のプロパティは、JMS への接続、ログ・ファイル名、エラー処理およびメッセージ出力を制御します。

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みます。

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。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。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
```

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 GoldenGate 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。

JMS 接続プロパティ

JMS 接続プロパティでは、JMS 統合用の JVM の起動方法など、接続を設定します。

jvm.boot options

ユーザー・イグジットで JVM を起動する場合に適用されるクラスパスおよび起動オプションを指定し

ます。パスには、UNIX/Linux の場合コロン (:)、Windows の場合セミコロン (;) の区切り文字が必要です。

構文は次のとおりです。

```
jvm.bootoptions={option}[, . . .]
```

オプションは、コマンドラインから実行される Java に渡されるものと同じです。これには、クラスパス、システム・プロパティ、使用されている Java のバージョンに対して有効な JVM メモリー・オプション (最大メモリー、初期メモリーなど) などがあります。有効なオプションは、JVM のバージョンおよびプロバイダによって異なります。

次に例を示します (すべて 1 行に記述します)。

```
jvm.bootoptions= -Djava.class.path=ggjava/ggjava.jar
                 -Dlog4j.configuration=my-log4j.properties
```

`log4j.configuration` プロパティには、log4j プロパティ・ファイルの完全修飾 URL を指定できます。デフォルトでは、このファイルはクラスパスで検索されます。独自の log4j 構成を使用することも、あらかじめ構成された log4j 設定である `log4j.properties`(デフォルト・レベルのロギング)、`debug_log4j.properties`(デバッグ・ロギング) または `trace_log4j.properties`(非常に詳細なロギング) のいずれかを使用することもできます。

jms.report.output

JMS レポートを書き込む場所を指定します。構文は次のとおりです。

```
jms.report.output=report|log|both
```

条件 : `report` は、JMS レポートを Oracle GoldenGate レポート・ファイルに送信します。これがデフォルトです。

`log` は、Java ログ・ファイル (構成されている場合) に書き込みます。

`both` は、両方の場所に送信します。

jms.report.time

レポート生成の頻度を時間に基づいて指定します。

```
jms.report.time={time-specification}
```

次の例では、30 秒ごと、45 分ごと、8 時間ごとにレポートを書き込みます。

```
jms.report.time=30sec
jms.report.time=45min
jms.report.time=8hr
```

jms.report.records

レポート生成の頻度をレコード数に基づいて指定します。

```
jms.report.records={number}
```

次の例では、1000 レコードごとにレポートを書き込みます。

```
jms.report.records=1000
```

jms.id

指定された形式の一意の識別子が JMS 統合からメッセージ取得 VAM に渡されることを指定します。これをレコードの一意のシーケンス ID として VAM で使用できます。

```
jms.id=ogg|time|wmq|activemq|{message-header}|{custom-java-class}
```

条件 : `ogg`: Oracle GoldenGate JMS 配信によって設定されるメッセージ・ヘッダー・プロパティ `GG_ID` を返します。

`time`: システム・タイムスタンプをメッセージ ID の開始点として使用します。

`wmq`: WebSphere MQ Message ID を VAM とともに使用するために再フォーマットします。

`activemq`: ActiveMQ Message ID を VAM とともに使用するために再フォーマットします。

{*message-header*}: ユーザーがカスタマイズした JMS メッセージ・ヘッダー (JMSMessageID、JMSCorrelationID、JMSTimestamp など) を含めるよう指定します。

{*custom-java-class*}: ID として使用される文字列を作成するカスタム Java クラスを指定します。

次に例を示します。

```
jms.id=time
jms.id=JMSMessageID
```

返される ID は、一意、増分かつ固定長です。数字が重複する場合、重複はスキップされます。メッセージ ID の長さを変更すると、Extract プロセスが異常終了します。

jms.destination

JNDI を介して検索されるキュー名またはトピック名を指定します。

```
jms.destination={jndi-name}
```

次に例を示します。

```
jms.destination=sampleQ
```

jms.connectionFactory

JNDI を介して検索される接続ファクトリ名を指定します。

```
jms.connectionFactory={jndi-name}
```

次に例を示します。

```
jms.connectionFactory=ConnectionFactory
```

jms.user、jms.password

JMS プロバイダで指定される、JMS 接続のユーザー名とパスワードを設定します。

```
jms.user={user-name}
jms.password={password}
```

これは、JNDI セキュリティに使用されません。JNDI 認証を設定するには、JNDI `java.naming.security` プロパティを参照してください。

次に例を示します。

```
jms.user=myuser
jms.password=mypasswd
```

JNDI プロパティ

メッセージ取得 VAM の特定のプロパティ以外に、JMS 統合で、接続ファクトリと宛先を検索するための初期コンテキストへの接続に必要な JNDI プロパティの設定もサポートされます。次のプロパティを設定する必要があります。

```
java.naming.provider.url={url}
java.naming.factory.initial={java-class-name}
```

JNDI セキュリティが有効な場合、次のプロパティを設定できます。

```
java.naming.security.principal={user-name}
java.naming.security.credentials={password-or-other-authenticator}
```

次に例を示します。

```
java.naming.provider.url= t3://localhost:7001
java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory
java.naming.security.principal=jndiuser
java.naming.security.credentials=jndipw
```

パーサー・プロパティ

プロパティで、各タイプのパーサー (固定、区切り、XML) のメッセージの形式および翻訳ルールを指定します。parser.type プロパティを設定して、使用するパーサーを指定します。残りのプロパティは、パーサー固有です。

パーサーのタイプの設定

次のプロパティはパーサー・タイプを設定します。

parser.type

使用するパーサーを指定します。

```
parser.type=fixed|delim|xml
```

条件 : `fixed` は、固定幅パーサーを起動します。

`delim` は、区切りパーサーを起動します。

`xml` は、XML パーサーを起動します。

次に例を示します。

```
parser.type=delim
```

固定パーサー・プロパティ

固定パーサーには次のプロパティが必要です。

fixed.schema

メッセージ取得のメタデータとして使用されるファイルのタイプを指定します。2 つの有効なオプションは、sourcedefs および copybook です。

```
fixed.schematyype=sourcedefs|copybook
```

次に例を示します。

```
fixed.schematype=copybook
```

このプロパティの値によって、受信データを正常に解析するために設定する必要がある他のプロパティが決まります。

fixed.sourcedefs

`fixed.schematype=sourcedefs` の場合、このプロパティで、使用されるソース定義ファイルの場所を指定します。

```
fixed.sourcedefs={file-location}
```

次に例を示します。

```
fixed.sourcedefs=dirdef/hrdemo.def
```

fixed.copybook

`fixed.schematype=copybook` の場合、このプロパティで、メッセージ取得プロセスで使用されるコピーブック・ファイルの場所を指定します。

```
fixed.copybook={file-location}
```

次に例を示します。

```
fixed.copybook=test_copy_book.cpy
```

fixed.header

データ・ブロック構造の決定に使用されるヘッダー情報を含む sourcedefs エントリまたは copybook レコードの名前を指定します。

```
fixed.header={record-name}
```

次に例を示します。

```
fixed.header=HEADER
```

fixed.seqid

各レコードを一意に識別するために使用される `seqid` を含むヘッダー・フィールドの名前、JMS プロパティまたはシステム値を指定します。この値は継続的に増分され、最後の文字が最下位である必要があります。

```
fixed.seqid={field-name}|${jms-property}|*seqid
```

条件： `field-name` は、seqid を含むヘッダー・フィールドの名前を示します。

`jms-property` は、指定された JMS ヘッダー・プロパティの値を使用します。この特別な値は `$jmsid` で、`jms.id` プロパティで選択されたメカニズムによって返される値を使用します。

seqid は、システムによって生成される、単純増分の 64 ビット整数を示します。

次に例を示します。

```
fixed.seqid=$jmsid
```

fixed.timestamp

タイムスタンプを含むフィールドの名前、JMS プロパティまたはシステム値を指定します。

```
fixed.timestamp={field-name}|${jms-property}|*ts
```

次に例を示します。

```
fixed.timestamp=TIMESTAMP
fixed.timestamp=$JMSTimeStamp
fixed.timestamp=*ts
```

fixed.timestamp.format

タイムスタンプ・フィールドの形式を指定します。

```
fixed.timestamp.format={format}
```

形式には、句読文字と次のものを含めることができます。

YYYY: 4 桁の年

YY: 2 桁の年

M[M]: 1 桁または 2 桁の月

D[D]: 1 桁または 2 桁の日

HH: 24 時間表記の時間

MI: 分

SS: 秒

Fn: n 個の端数

デフォルトの形式は、"YYYY-MM-DD:HH:MI:SS.FFF" です。

次に例を示します。

```
fixed.timestamp.format=YYYY-MM-DD-HH.MI.SS
```

fixed.txid

トランザクションを一意に識別するために使用される txid を含むフィールドの名前、JMS プロパティまたはシステム値を指定します。この値は、トランザクションごとに増分されます。

```
fixed.txid={field-name}|${jms-property}|*txid
```

ほとんどの場合、システム値 *txid を使用することが推奨されます。

次に例を示します。

```
fixed.txid=$JMSTxId
fixed.txid=*txid
```

fixed.txowner

トランザクションに関連付けられているユーザー名を含むフィールドの名前、JMS プロパティまたは静的値を指定します。この値は、特定のトランザクションを処理から除外する場合に使用できます。これは、オプションのプロパティです。

```
fixed.txowner={field-name}|${jms-property}|"{value}"
```

次に例を示します。

```
fixed.txowner=$MessageOwner
fixed.txowner="jsmith"
```

fixed.txname

トランザクションに関連付けられている任意の名前を含むフィールドの名前、JMS プロパティまたは静的値を指定します。これは、オプションのプロパティです。

```
fixed.txname={field-name}|${jms-property}|"{value}"
```

次に例を示します。

```
fixed.txname="fixedtx"
```

fixed.optype

操作タイプを含むフィールドの名前または JMS プロパティを指定します。これは、後述の項で指定される `fixed.optype` 値に対して検証されます。

```
fixed.header.optype={field-name}|${jms-property}
```

次に例を示します。

```
fixed.header.optype=FUNCTION
```

fixed.optype.insertval

この値は、挿入操作を識別します。デフォルトは、`I` です。

```
fixed.optype.insertval={value}|\x{hex-value}
```

次に例を示します。

```
fixed.optype.insertval=A
```

fixed.optype.updateval

この値は、更新操作を識別します。デフォルトは、`U` です。

```
fixed.optype.updateval={value}|\x{hex-value}
```

次に例を示します。

```
fixed.optype.updateval=M
```

fixed.optype.deleteval

この値は削除操作を識別します。デフォルトは、`D` です。

```
fixed.optype.deleteval={value}|\x{hex-value}
```

次に例を示します。

```
fixed.optype.deleteval=R
```

fixed.table

表の名前を指定します。これによって、ヘッダー以外のデータ部分の翻訳に必要なデータ・レコード定義をパーサーが検索できます。

```
fixed.table={field-name}|${jms-property}[, . . .]
```

カンマ区切りの複数のフィールド名を使用して表の名前を決定できます。各フィールド名は、`fixed.header` プロパティまたは JMS プロパティによって定義されるヘッダー・レコード内のフィールドに対応します。これらのフィールドの値は連結されてデータ・レコードを識別します。

次に例を示します。

```
fixed.table=$JMSTableName
fixed.table=SOURCE_Db,SOURCE_Db_Rec_Version
```

fixed.schema

SCHEMA.TABLE 表名を生成する際のスキーマの静的な名前を指定します。

```
fixed.schema="{value}"
```

次に例を示します。

```
fixed.schema="OGG"
```

fixed.txind

トランザクション・インジケータ値に対して検証される、トランザクション・インジケータを含むフィールドの名前または JMS プロパティを指定します。これが定義されていない場合、1 つのメッセージ内のすべての操作が 1 つのトランザクション内で発生したとみなされます。定義されている場合、これによってトランザクションの始まり、中間および終わりが決まります。このように定義されたトランザクションは複数メッセージにわたることができます。これは、オプションのプロパティです。

```
fixed.txind={field-name}|${jms-property}
```

次に例を示します。

```
fixed.txind=$TX_IND
```

fixed.txind.beginval

この値は、操作をトランザクションの始まりとして識別します。デフォルトは、`B` です。

```
fixed.txind.beginval={value}|\x{hex-value}
```

次に例を示します。

```
fixed.txind.beginval=0
```

fixed.txind.middleval

この値は、操作をトランザクションの中間として識別します。デフォルトは、`M` です。

```
fixed.txind.middleval={value}|\x{hex-value}
```

次に例を示します。

```
fixed.txind.middleval=1
```

fixed.txind.endval

この値は、操作をトランザクションの終わりとして識別します。デフォルトは、*E* です。

```
fixed.txind.endval={value}|\x{hex-value}
```

次に例を示します。

```
fixed.txind.endval=2
```

fixed.txind.wholeval

この値は、操作をトランザクション全体として識別します。デフォルトは、*W* です。

```
fixed.txind.wholeval={value}|\x{hex-value}
```

次に例を示します。

```
fixed.txind.wholeval=3
```

区切りパーサー・プロパティ

特に記載のないかぎり、区切りパーサーには次のプロパティが必要です。

delim.sourcedefs

使用するソース定義ファイルの場所を指定します。

```
delim.sourcedefs={file-location}
```

次に例を示します。

```
delim.sourcedefs=dirdef/hrdemo.def
```

delim.header

データの前に配置される値のリストを指定し、各々に名前を割り当てます。

```
delim.header={name},[. . .]
```

名前は一意である必要があります。これらは、他の `delim` プロパティまたはヘッダー・フィールドが使用される場合に参照されます。

次に例を示します。

```
delim.header=optype, tablename, ts
delim.timestamp=ts
```

delim.seqid

各レコードを一意に識別するために使用される `seqid` を含むヘッダー・フィールドの名前、JMS プロパティまたはシステム値を指定します。この値は増分され、最後の文字が最下位である必要があります。

```
delim.seqid={field-name}|${jms-property}|*seqid
```

条件: `field-name` は、seqid を含むヘッダー・フィールドの名前を示します。

`jms-property` では、指定された JMS ヘッダー・プロパティの値を使用します。この特別な値は $jmsid で、jms.id プロパティで選択されたメカニズムによって返される値を使用します。

`seqid` は、システムによって生成される、単純で継続的に増分される 64 ビット整数を示します。

次に例を示します。

```
delim.seqid=$jmsid
```

delim.timestamp

タイムスタンプを含む JMS プロパティの名前、ヘッダー・フィールドまたはシステム値を指定します。

```
delim.timestamp={field-name}|${jms-property}|*ts
```

次に例を示します。

```
delim.timestamp=TIMESTAMP
delim.timestamp=$JMSTimeStamp
delim.timestamp=*ts
```

delim.timestamp.format

タイムスタンプ・フィールドの形式を指定します。

```
delim.timestamp.format={format}
```

形式には、句読文字と次のものを含めることができます。

`YYYY`: 4 桁の年

`YY`: 2 桁の年

`M[M]`: 1 桁または 2 桁の月

`D[D]`: 1 桁または 2 桁の日

`HH`: 24 時間表記の時間

`MI`: 分

`SS`: 秒

`Fn`: n 個の端数

デフォルトの形式は、"`YYYY-MM-DD:HH:MI:SS.FFF`" です。

次に例を示します。

```
delim.timestamp.format=YYYY-MM-DD-HH.MI.SS
```

delim.txid

トランザクションを一意に識別するために使用される `txid` を含む JMS プロパティの名前、ヘッダー・フィールドまたはシステム値を指定します。この値は、トランザクションごとに増分されます。

```
delim.txid={field-name}|${jms-property}|*txid
```

ほとんどの場合、システム値 `*txid` を使用することが推奨されます。

次に例を示します。

```
delim.txid=$JMSTxId
delim.txid=*txid
```

delim.txowner

トランザクションに関連付けられているユーザー名を含む JMS プロパティの名前、ヘッダー・フィールドまたは静的値を指定します。この値は、特定のトランザクションを処理から除外する場合に使用できます。これは、オプションのプロパティです。

```
delim.txowner={field-name}|${jms-property}|"{value}"
```

次に例を示します。

```
delim.txowner=$MessageOwner
delim.txowner="jsmith"
```

delim.txname

トランザクションに関連付けられている任意の名前を含む JMS プロパティの名前、ヘッダー・フィールドまたは静的値を指定します。これは、オプションのプロパティです。

```
delim.txname={field-name}|${jms-property}|"{value}"
```

次に例を示します。

```
delim.txname="fixedtx"
```

delim.optype

操作タイプを含む JMS プロパティまたはヘッダー・フィールドの名前を指定します。これは、`delim.optype.insertval`、`delim.optype.updateval` および `delim.optype.deleteval` の値と比較されて、操作が決まります。

```
delim.optype={field-name}|${jms-property}
```

次に例を示します。

```
delim.optype=optype
```

delim.optype.insertval

この値は、挿入操作を識別します。デフォルトは、I です。

```
delim.optype.insertval={value}|\x{hex-value}
```

次に例を示します。

```
delim.optype.insertval=A
```

delim.optype.updateval

この値は、更新操作を識別します。デフォルトは、U です。

```
delim.optype.updateval={value}|\x{hex-value}
```

次に例を示します。

```
delim.optype.updateval=M
```

delim.optype.deleteval

この値は、挿入操作を識別します。デフォルトは、D です。

```
delim.optype.deleteval={value}|\x{hex-value}
```

次に例を示します。

```
delim.optype.deleteval=R
```

delim.schemaandtable

SCHEMA.TABLE 形式でスキーマおよび表の名前を含む JMS プロパティまたはヘッダー・フィールドの名前を指定します。

```
delim.schemaandtable={field-name}|${jms-property}
```

次に例を示します。

```
delim.schemaandtable=$FullTableName
```

delim.schema

スキーマ名を含む JMS プロパティの名前、ヘッダー・フィールドまたはハードコードされた値を指定します。

```
delim.schema={field-name}|${jms-property}|"{value}"
```

次に例を示します。

```
delim.schema="OGG"
```

delim.table

表名を含む JMS プロパティまたはヘッダー・フィールドの名前を指定します。

```
delim.table={field-name}|${jms-property}
```

次に例を示します。

```
delim.table=TABLE_NAME
```

delim.txind

beginval、middleval、endval または wholeval に対して検証される、トランザクション・インジケータを含む JMS プロパティまたはヘッダー・フィールドの名前を指定します。このプロパティが設定されていない場合、1 つのメッセージ内のすべての操作が 1 つのトランザクション内にあるとみなされます。定義されている場合、これによってトランザクションの始まり、中間および終わりが決まります。このように定義されたトランザクションは複数メッセージにわたることができます。これは、オプションのプロパティです。

```
delim.txind={field-name}|${jms-property}
```

次に例を示します。

```
delim.txind=txind
```

delim.txind.beginval

操作をトランザクションの始まりとして識別する値。デフォルトは、B です。

```
delim.txind.beginval={value}|\x{hex-value}
```

次に例を示します。

```
delim.txind.beginval=0
```

delim.txind.middleval

操作をトランザクションの中間として識別する値。デフォルトは、M です。

```
delim.txind.middleval={value}|\x{hex-value}
```

次に例を示します。

```
delim.txind.middleval=1
```

delim.txind.endval

操作をトランザクションの終わりとして識別する値。デフォルトは、E です。

```
delim.txind.endval={value}|\x{hex-value}
```

次に例を示します。

```
delim.txind.endval=2
```

delim.txind.wholeval

操作をトランザクション全体として識別する値。デフォルトは、W です。

```
delim.txind.wholeval={value}|\x{hex-value}
```

次に例を示します。

```
delim.txind.wholeval=3
```

delim.fielddelim

データ内のフィールド (列) の区切りに使用されるデリミタ値を指定します。この値は、文字または 16 進値を使用して定義されます。

```
delim.fielddelim={value}|\x{hex-value}
```

次に例を示します。

```
delim.fielddelim=,
delim.fielddelim=\xc7
```

delim.linedelim

データ内の行 (レコード) の区切りに使用されるデリミタ値を指定します。この値は、文字または 16 進値を使用して定義されます。

```
delim.linedelim={value}|\x{hex-value}
```

次に例を示します。

```
delim.linedelim=||
delim.linedelim=\x0a
```

delim.quote

引用符付きのデータの識別に使用される値を指定します。この値は、文字または 16 進値を使用して定義されます。

```
delim.quote={value}|\x{hex-value}
```

次に例を示します。

```
delim.quote="
```

delim.nullindicator

NULL データの識別に使用される値を指定します。この値は、文字または 16 進値を使用して定義されます。

```
delim.nullindicator={value}|\x{hex-value}
```

次に例を示します。

```
delim.nullindicator=NULL
```

delim.fielddelim.escaped

データ内に本来のフィールド・デリミタが存在することを示す値を指定します。このフィールド・デリミタは、`fielddelim.escaped` 値で置き換えられます。

```
delim.fielddelim.escaped={value}|\x{hex-value}
```

次の例では、カンマをフィールド・デリミタとし、$ エスケープ文字で囲んで指定します。

```
delim.fielddelim.escaped=$,$
```

delim.linedelim.escaped

データ内に本来の行デリミタが存在することを示す値を指定します。この行デリミタは、`linedelim.escaped` 値で置き換えられます。

```
delim.linedelim.escaped={value}|\x{hex-value}
```

次に例を示します。

```
delim.linedelim.escaped=\x0affa0
```

delim.quote.escaped

データ内に本来の引用符が存在することを示す値を指定します。この引用府の値は、`quote.escaped` 値で置き換えられます。

```
delim.quote.escaped={value}|\x{hex-value}
```

次に例を示します。

```
delim.quote.escaped=""
```

delim.nullindicator.escaped

データ内に本来の null インジケータが存在することを示す値を指定します。このインジケータは、nullindicator.escaped 値で置き換えられます。

```
delim.nullindicator.escaped={value}|\x{hex-value}
```

次に例を示します。

```
delim.nullindicator.escaped={NULL}
```

delim.hasbefores

データ内にビフォア値が存在するかどうかを指定します。

```
delim.hasbefores=true|false
```

デフォルトは、false です。delim.hasbefores が true に設定されている場合、パーサーは、すべてのレコードについて列のビフォア値とアフター値があると想定します。ビフォア値は更新と削除に使用され、アフター値は更新と挿入に使用されます。afterfirst プロパティは、ビフォア・イメージがアフター・イメージの前か後かを指定します。delim.hasbefores が false の場合、ビフォア値は想定されません。

次に例を示します。

```
delim.hasbefores=true
```

delim.hasnames

データ内に列名が存在するかどうかを指定します。

```
delim.hasnames=true|false
```

デフォルトは、false です。true の場合、パーサーはすべてのレコードについて列名があると想定します。パーサーは、想定される列名に対して列名を検証します。false の場合、列名は想定されません。

次に例を示します。

```
delim.hasnames=true
```

delim.afterfirst

アフター値がビフォア値の前に配置されるか、後に配置されるかを指定します。

```
delim.afterfirst=true|false
```

デフォルトは、false です。true の場合、パーサーは、アフター値はビフォア値の前にあると想定します。false の場合、アフター値はビフォア値の前にあります。

次に例を示します。

```
delim.afterfirst=true
```

delim.isgrouped

すべての列の列名、ビフォア・イメージおよびアフター・イメージがグループ化されるか、列ごとに順に配置されるかを指定します。

```
delim.isgrouped=true|false
```

デフォルトは、false です。true の場合、パーサーは、列名のグループ (hasnames が true の場合)、次にビフォア値のグループ (hasbefores の場合)、その次にアフター値のグループがある (afterfirst 設定によってはビフォア値とアフター値の順序が逆になる) と想定します。false の場合、パーサーは、列名 (hasnames の場合)、ビフォア値 (hasbefores の場合) およびアフター値が列ごとにあると想定します。

次に例を示します。

```
delim.isgrouped=true
```

delim.dateformat

列データの日付形式を指定します。日付の解析に使用される形式は、parser.timestamp.format に使用される形式のサブセットです。これは、グローバル・レベル、表レベルまたは列レベルで指定されます。

```
delim.dateformat={format}
delim.dateformat.{TABLE}={format}
delim.dateformat.{TABLE}.{COLUMN}={format}
```

条件 : format は、parser.timestamp.format に定義された形式です。

TABLE は、完全修飾表名です。

COLUMN は、指定された表の列です。

次に例を示します。

```
delim.dateformat=YYYY-MM-DD HH:MI:SS
delim.dateformat.MY.TABLE=DD/MM/YY-HH.MI.SS
delim.dateformat.MY.TABLE.EXP_DATE=YYMM
```

XML パーサー・プロパティ

XML パーサーには次のプロパティが必要です。

xml.sourcedefs

ソース定義ファイルの場所を指定します。

```
xml.sourcedefs={file-location}
```

次に例を示します。

```
xml.sourcedefs=dirdef/hrdemo.def
```

xml.rules

メッセージの解析とトランザクション、操作および列の変換の XML ルールのリストを指定します。

```
xml.rules={xml-rule-name}[, . . .]
```

指定された XML ルールは、リストされた順に処理されます。特定の XML ドキュメントに一致するすべてのルールによって、トランザクション、操作および列を作成できます。指定される XML ルールは、トランザクション・タイプまたは操作タイプのルールである必要があります。

次に例を示します。

```
xml.rules=dyntxrule, statoprule
```

{rulename}.type

XML ルールのタイプを指定します。

```
{rulename}.type=tx|op|col
```

条件： `tx` は、トランザクション・ルールを示します。

`op` は、操作ルールを示します。

`col` は、列ルールを示します。

次に例を示します。

```
dyntxrule.type=tx
statoprule.type=op
```

{rulename}.match

ルールが特定のドキュメントに対してアクティブ化されるかどうかの決定に使用される XPath 式を指定します。

```
{rulename}.match={xpath-expression}
```

XPath 式がドキュメントからノードを返す場合、ルールに一致され、後続の処理が行われます。ノードを返さない場合、そのドキュメントに対してルールは無視されます。

次の例では、ドキュメントにルート要素 `Transaction` がある場合、`dyntxrule` がアクティブ化されます。

```
dyntxrule.match=/Transaction
```

`statoprule` は `stattxtule` のサブルールです。次の例では、親ルールのマッチング・ノードに子要素 `NewMyTableEntry` がある場合、`statoprule` がアクティブ化されます。

```
statoprule.match=./NewMyTableEntry
```

{rulename}.subrules

親ルールが一致によってアクティブ化された場合、一致をチェックするルール名のリストを指定します。

```
{rulename}.subrules={xml-rule-name}[, . . .]
```

指定された XML ルールは、リストされた順に処理されます。一致するすべてのルールによって、トランザクション、操作および列を作成できます。

有効なサブルールは、親タイプによって決まります。トランザクション・ルールは、操作サブルールのみ持つことができます。操作ルールは、操作または列サブルールを持つことができます。列ルールは、サブルールを持つことはできません。

次に例を示します。

```
dyntxrule.subrules=dynoprule
statoprule.subrules=statkeycolrule, statcol1rule
```

{txrule}.timestamp

1) 指定された XPath 式または JMS プロパティに含まれたトランザクション・コミット・タイムスタンプの使用、または 2) 現在のシステム時間の使用をアダプタに指定することで、トランザクションのタイムスタンプを制御します。これは、オプションのプロパティです。

```
{txrule}.timestamp={xpath-expression}|${jms-property}|*ts
```

トランザクションのタイムスタンプは操作レベルでオーバーライドすることも、操作レベルでのみ存在することもできます。XPath 式は、`@att`、`text()` などの値アクセッサで終わる必要があります。

次に例を示します。

```
dyntxrule.timestamp=@ts
```

{txrule}.timestamp.format

タイムスタンプ・フィールドの形式を指定します。

```
{txrule}.timestamp.format={format}
```

形式には、句読文字と次のものを含めることができます。

`YYYY`: 4 桁の年

`YY`: 2 桁の年

`M[M]`: 1 桁または 2 桁の月

`D[D]`: 1 桁または 2 桁の日

`HH`: 24 時間表記の時間

`MI`: 分

`SS`: 秒

`Fn`: n 個の端数

デフォルトの形式は、"`YYYY-MM-DD:HH:MI:SS.FFF`" です。

次に例を示します。

```
dyntxrule.timestamp.format=YYYY-MM-DD-HH.MI.SS
```

{txrule}.seqid

特定のトランザクションの `seqid` を指定します。これは、各メッセージに複数のトランザクションがある場合に使用できます。トランザクションの `seqid` を含む XPath 式、JMS プロパティまたはシステム値を決定します。XPath 式は、`@att`、`text()` などの値アクセッサで終わる必要があります。

```
{txrule}.seqid={xpath-expression}|${jms-property}|*seqid
```

次に例を示します。

```
dyntxrule.seqid=@seqid
```

{txrule}.txid

トランザクションを一意に識別するために使用される `txid` を含む XPath 式、JMS プロパティまたはシステム値を指定します。この値は、トランザクションごとに増分されます。

```
{txrule}.txid={xpath-expression}|${jms-property}|*txid
```

ほとんどの場合、システム値 `*txid` を使用することが推奨されます。

次に例を示します。

```
dyntxrule.txid=$JMSTxId
dyntxrule.txid=*txid
```

{txrule}.txowner

トランザクションに関連付けられている任意のユーザー名を含む XPath 式、JMS プロパティまたは静的値を指定します。この値は、特定のトランザクションを処理から除外する場合に使用できます。

```
{txrule}.txowner={xpath-expression}|${jms-property}|"{value}"
```

次に例を示します。

```
dyntxrule.txowner=$MessageOwner
dyntxrule.txowner="jsmith"
```

{txrule}.txname

トランザクションに関連付けられている任意の名前を含む XPath 式、JMS プロパティまたは静的値を指定します。これは、オプションのプロパティです。

```
{txrule}.txname={xpath-expression}|${jms-property}|"value"
```

次に例を示します。

```
dyntxrule.txname="fixedtx"
```

{oprule}.timestamp

1) 指定された XPath 式または JMS プロパティに含まれたトランザクション・コミット・タイムスタンプの使用、または 2) 現在のシステム時間の使用をアダプタに指定することで、操作のタイムスタンプを制御します。これは、オプションのプロパティです。

```
{oprule}.timestamp={xpath-expression}|${jms-property}|*ts
```

操作のタイムスタンプは、トランザクション・レベルのタイムスタンプをオーバーライドします。

XPath 式は、`@att`、`text()` などの値アクセッサで終わる必要があります。

次に例を示します。

```
statoprule.timestamp=./CreateTime/text()
```

{oprule}.timestamp.format

タイムスタンプ・フィールドの形式を指定します。

```
{oprule}.timestamp.format={format}
```

形式には、句読文字と次のものを含めることができます。

`YYYY`: 4 桁の年

`YY`: 2 桁の年

`M[M]`: 1 桁または 2 桁の月

`D[D]`: 1 桁または 2 桁の日

`HH`: 24 時間表記の時間

MI: 分

SS: 秒

Fn: n 個の端数

デフォルトの形式は、"YYYY-MM-DD:HH:MI:SS.FFF" です。

次に例を示します。

```
statoprule.timestamp.format=YYYY-MM-DD-HH.MI.SS
```

{oprule}.seqid

特定の操作の seqid を指定します。操作の seqid を含む XPath 式、JMS プロパティまたはシステム値を使用します。これは、親トランザクション・ルールで定義された seqid をオーバーライドします。親トランザクション・ルールがない場合、必要です。これは、オプションのプロパティです。

XPath 式は、@att、text() などの値アクセッサで終わる必要があります。

```
{oprule}.seqid={xpath-expression}|${jms-property}|*seqid
```

次に例を示します。

```
dynoprule.seqid=@seqid
```

{oprule}.txid

トランザクションを一意に識別するために使用される txid を含む XPath 式、JMS プロパティまたはシステム値を指定します。これは、親トランザクション・ルールで定義された txid をオーバーライドし、親トランザクション・ルールがない場合に必要です。この値は、トランザクションごとに増分されます。これは、オプションのプロパティです。

```
{oprule}.txid={xpath-expression}|${jms-property}|*txid
```

ほとんどの場合、システム値 *txid を使用することが推奨されます。

次に例を示します。

```
dynoprule.txid=$JMSTxId
dynoprule.txid=*txid
```

{oprule}.txowner

トランザクションに関連付けられている任意のユーザー名を含む XPath 式、JMS プロパティまたは静的値を指定します。この値は、特定のトランザクションを処理から除外する場合に使用できます。これは、オプションのプロパティです。

```
{oprule}.txowner={xpath-expression}|${jms-property}|"{value}"
```

次に例を示します。

```
dynoprule.txowner=$MessageOwner
dynoprule.txowner="jsmith"
```

{oprule}.txname

トランザクションに関連付けられている任意の名前を含む XPath 式、JMS プロパティまたは静的値を指定します。これは、オプションのプロパティです。

```
{oprule}.txname={xpath-expression}|${jms-property}|"{value}"
```

次に例を示します。

```
dynoprule.txname="fixedtx"
```

{oprule}.schemandtable

SCHEMA.TABLE 形式でスキーマおよび表の名前を含む XPath 式、JMS プロパティまたはハードコードされた値を指定します。XPath 式は、@att、text() などの値アクセッサで終わる必要があります。値は、表がソース定義に存在するかどうか検証されます。

```
{oprule}.schemaandtable={xpath-expression}|${jms-property}|"{value}"
```

次に例を示します。

```
statoprule.schemaandtable="MY.TABLE"
```

{oprule}.schema

スキーマ名を含む XPath 式、JMS プロパティまたはハードコードされた値を指定します。XPath 式は、@att、text() などの値アクセッサで終わる必要があります。

```
{oprule}.schema={xpath-expression}|${jms-property}|"value"
```

次に例を示します。

```
statoprule.schema=@schema
```

{oprule}.table

表名を含む XPath 式、JMS プロパティまたはハードコードされた値を指定します。XPath 式は、@att、text() などの値アクセッサで終わる必要があります。

```
{oprule}.table={xpath-expression}|${jms-property}|"value"
```

次に例を示します。

```
statoprule.table=$TableName
```

{oprule}.optype

optype insertval などに対して検証される optype を含む XPath 式、JMS プロパティまたはリテラル値を指定します。XPath 式は、@att、text() などの値アクセッサで終わる必要があります。

```
{oprule}.optype={xpath-expression}|${jms-property}|"value"
```

次に例を示します。

```
dynoprule.optype=@type
statoprule.optype="I"
```

{oprule}.optype.insertval

挿入操作を識別する値を指定します。デフォルトは、I です。

```
{oprule}.optype.insertval={value}|\x{hex-value}
```

次に例を示します。

```
dynoprule.optype.insertval=A
```

{oprule}.optype.updateval

更新操作を識別する値を指定します。デフォルトは、U です。

```
{oprule}.optype.updateval={value}|\x{hex-value}
```

次に例を示します。

```
dynoprule.optype.updateval=M
```

{oprule}.optype.deleteval

削除操作を識別する値を指定します。デフォルトは、D です。

```
{oprule}.optype.deleteval={value}|\x{hex-value}
```

次に例を示します。

```
dynoprule.optype.deleteval=R
```

{oprule}.txind

beginval またはトランザクション内の位置を識別する他の値に対して検証されるトランザクション・インジケータを含む XPath 式または JMS プロパティを指定します。このプロパティが定義されていない場合、1 つのメッセージ内のすべての操作が 1 つのトランザクション内で発生したとみなされます。トランザクションの始まり、中間および終わりを指定します。XPath 式は、@att、text() などの値アクセッサで終わる必要があります。このように定義されたトランザクションは複数メッセージにわたることができます。これは、オプションのプロパティです。

```
{oprule}.txind={xpath-expression}|${jms-property}
```

次に例を示します。

```
dynoprule.txind=@txind
```

{oprule}.txind.beginval

操作をトランザクションの始まりとして識別する値を指定します。デフォルトは、B です。

```
{oprule}.txind.beginval={value}|\x{hex-value}
```

次に例を示します。

```
dynoprule.txind.beginval=0
```

{oprule}.txind.middleval

操作をトランザクションの中間として識別する値を指定します。デフォルトは、M です。

```
{oprule}.txind.middleval={value}|\x{hex-value}
```

次に例を示します。

```
dynoprule.txind.middleval=1
```

{oprule}.txind.endval

操作をトランザクションの終わりとして識別する値を指定します。デフォルトは、E です。

```
{oprule}.txind.endval={value}|\x{hex-value}
```

次に例を示します。

```
dynoprule.txind.endval=2
```

{oprule}.txind.wholeval

操作をトランザクション全体として識別する値を指定します。デフォルトは、W です。

```
{oprule}.txind.wholeval={value}|\x{hex-value}
```

次に例を示します。

```
dynoprule.txind.wholeval=3
```

{colrule}.name

列名を含む XPath 式またはハードコードされた値を指定します。これを指定しない場合、列索引を指定する必要があります。列索引から列名が解決されます。指定された場合、列名がソース定義ファイルに対して検証されます。XPath 式は、`@att`、`text()` などの値アクセッサで終わる必要があります。

```
{colrule}.name={xpath-expression}|"value"
```

次に例を示します。

```
dyncolrule.name=@name
statkeycolrule.name="keycol"
```

{colrule}.index

列索引を含む XPath 式またはハードコードされた値を指定します。これを指定しない場合、列名を指定する必要があります。列名から列索引が解決されます。指定された場合、列索引がソース定義ファイルに対して検証されます。XPath 式は、`@att`、`text()` などの値アクセッサで終わる必要があります。

```
{colrule}.index={xpath-expression}|"value"
```

次に例を示します。

```
dyncolrule.index=@index
statkeycolrule.index=1
```

{colrule}.value

列値を含む XPath 式またはハードコードされた値を指定します。XPath 式は、`@att`、`text()` などの値アクセッサで終わる必要があります。ノードまたは属性が存在せず、XPath 式が値を返さない場合、列値は null とみなされます。null と欠落値 (更新に対する) を区別するために、`isnull` および `ismissing` プロパティを設定する必要があります。返される値は、ビフォア値の削除およびアフター値の更新 / 挿入に使用されます。

```
{colrule}.value={xpath-expression}|"value"
```

次に例を示します。

```
statkeycolrule.value=./text()
```

{colrule}.isnull

列値が null かどうかの検出に使用される XPath 式を指定します。XPath 式は、`@att`、`text()` などの値アクセッサで終わる必要があります。XPath 式が値を返す場合、列値は null です。これは、オプションのプロパティです。

```
{colrule}.isnull={xpath-expression}
```

次に例を示します。

```
dyncolrule.isnull=@isnull
```

{colrule}.ismissing

列値が欠落しているかどうかの検出に使用される XPath 式を指定します。XPath 式は、`@att`、`text()` などの値アクセッサで終わる必要があります。XPath 式が値を返す場合、列値は欠落しています。これは、オプションのプロパティです。

```
{colrule}.ismissing={xpath-expression}
```

次に例を示します。

```
dyncolrule.ismissing=./missing
```

{colrule}.before.value

`{colrule}.value` をオーバーライドし、更新または削除に使用されるビフォア値の取得方法を明示的に指定します。この形式は、`{colrule}.value` と同じです。これは、オプションのプロパティです。

次に例を示します。

```
dyncolrule.before.value=./beforevalue/text()
```

{colrule}.before.isnull

`{colrule}.isnull` をオーバーライドし、更新または削除用のビフォア値が null かどうかの決定方法を明示的に指定します。この形式は、`{colrule}.isnull` と同じです。これは、オプションのプロパティです。

次に例を示します。

```
dyncolrule.before.isnull=./beforevalue/@isnull
```

{colrule}.before.ismissing

`{colrule}.ismissing` をオーバーライドし、更新または削除用のビフォア値が欠落しているかどうかの決定方法を明示的に指定します。この形式は、`{colrule}.ismissing` と同じです。これは、オプションのプロパティです。

次に例を示します。

```
dyncolrule.before.ismissing=./beforevalue/missing
```

{colrule}.after.value

{*colrule*}.value をオーバーライドし、更新または削除に使用されるアフター値の取得方法を明示的に指定します。この形式は、{*colrule*}.value と同じです。これは、オプションのプロパティです。

次に例を示します。

```
dyncolrule.after.value=./aftervalue/text()
```

*{colrule}.after.isnull

{*colrule*}.isnull をオーバーライドし、更新または削除用のアフター値が null かどうかの決定方法を明示的に指定します。この形式は、{*colrule*}.isnull と同じです。これは、オプションのプロパティです。

次に例を示します。

```
dyncolrule.after.isnull=./aftervalue/@isnull
```

{colrule}.after.ismissing

{*colrule*}.ismissing をオーバーライドし、更新または削除用のアフター値が欠落しているかどうかの決定方法を明示的に指定します。この形式は、{*colrule*}.ismissing と同じです。これは、オプションのプロパティです。

次に例を示します。

```
dyncolrule.after.ismissing=./aftervalue/missing
```

第 6 章

UE: メッセージ配信の構成

メッセージの配信用にアダプタを構成するには、ユーザー・イグジット・プロパティ・ファイルでプロパティを設定して、データ・ポンプとしてユーザー・イグジットを実行するよう Extract を構成し、使用する組込みまたはカスタムのイベント・ハンドラを識別する必要があります。

ユーザー・イグジット・プロパティ・ファイルでの JRE の構成

Java 用 Oracle GoldenGate メイン jar (ggjava.jar) の場所を指すようユーザー・イグジット・プロパティ・ファイルを変更します。必要に応じて追加の JVM ランタイム起動オプション (起動時に JVM に直接渡される) を設定します。

```
javawriter.bootoptions=-Djava.class.path=ggjava/ggjava.jar
-Dlog4j.configuration=log4j.properties -Xmx512m
```

特に次のオプションに注意してください。

- java.class.path には、コア・アプリケーション (ggjava.jar) 以外にカスタム jar を含めることができます。現在のディレクトリ (.) がデフォルトでクラスパスに含められます。Oracle GoldenGate インストール・ディレクトリを基準にファイルを参照し、Java プロパティ・ファイル、Velocity テンプレートおよび dirprm ディレクトリ内の他のクラスパス・リソースを格納できます。Java アプリケーション・プロパティ・ファイル内のクラスパスに追加することもできます。
- log4j.configuration オプションは、クラスパス内にある log4j プロパティ・ファイルを指定します。基本ロギング (log4j.properties)、デバッグ・ロギング (debug-log4j.properties) および詳細トレースレベル・ロギング (trace-log4j.properties) 用にあらかじめ構成されたデフォルトの log4j 設定が resources/classes ディレクトリにあります。

システム用にユーザー・イグジット・プロパティ・ファイルを正しく構成したら、通常変更しません。構成オプションの詳細は、71 ページの「ユーザー・イグジット・プロパティ」を参照してください。

ユーザー・イグジットを実行するためのデータ・ポンプの構成

ユーザー・イグジット Extract は、データ・ポンプとして構成されます。データ・ポンプはローカル証跡 (`dirdat/aa` など) を消費し、データをユーザー・イグジットに送信します。ユーザー・イグジットは、すべてのデータの処理に使用されます。

データ・ポンプ Extract の追加例は、次のとおりです。

```
ADD EXTRACT javaue, EXTTRAILSOURCE ./dirdat/aa
```

前述の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 パラメータ・ファイル (javaue.prm) で、ユーザー・イグジット・ライブラリの場所を指定します。

表 2　　ユーザー・イグジット Extract のパラメータ

パラメータ	説明
`EXTRACT javaue`	すべての Extract パラメータ・ファイルは Extract 名で始まります。
`SOURCEDEFS ./dirdef/tcust.def`	Extract プロセスは、証跡データを表すメタデータを必要とします。これは、データベースまたはソース定義ファイルから取得します。このメタデータは、読み取る対象の証跡 (./dirdat/aa) の列名およびデータ型を定義します。
`SETENV (GGS_USEREXIT_CONF = "dirprm/javaue.properties")`	(オプション)C ユーザー・イグジット・ライブラリのプロパティ・ファイルの絶対パスまたは相対 (Extract 実行可能ファイルを基準とした) パス。デフォルト値は、Extract と同じディレクトリの javawriter.properties です。
`SETENV (GGS_JAVAUSEREXIT_CONF = "dirprm/javaue.properties")`	(オプション)Java プロパティ。この例では、dirpm ディレクトリにプロパティ・ファイルを配置します。
`CUSEREXIT ggjava_ue.dll CUSEREXIT PASSTHRU INCLUDEUPDATEBEFORES`	`CUSEREXIT` パラメータには、次のものを含めます。 ◆ ユーザー・イグジット・ライブラリの場所。UNIX の場合、ライブラリには .so 接尾辞が付きます。 ◆ `CUSEREXIT`: コールバック関数名。大文字である必要があります。 ◆ `PASSTHRU`: ダミー・ターゲット証跡の必要がなくなります。 ◆ `INCLUDEUPDATEBEFORES`: トランザクションの整合性のために必要です。
`TABLE schema.*;`	ユーザー・イグジットに渡す表。含まれていない表はスキップされます。*フィルタなし*をユーザー・イグジット Extract で行えます。それ以外の場合、トランザクション・マーカーは省略されます。プライマリ Extract でのフィルタ、別のアップストリーム・データ・ポンプの使用または Java アプリケーションでのデータの直接のフィルタを行えます。

前述の 2 つの環境プロパティはオプションですが、有用です。たとえば、これらによって、すべてのプロパティ・ファイルをデフォルトの場所ではなく、dirprm プロパティに配置できます。

- `SETENV (GGS_USEREXIT_CONF = "dirprm/javaue.properties")`

  これは、ユーザー・イグジット共有ライブラリに使用されるデフォルトの構成ファイルを変更します。指定される値は、絶対パスまたは Extract(または Replicat) を基準としたパスです。使用されるデフォルト・ファイルは、Extract と同じディレクトリの javawriter.properties です。前述の例では、相対パスを使用してこのプロパティ・ファイルを dirprm ディレクトリに配置します。

- `SETENV (GGS_JAVAUSEREXIT_CONF = "dirprm/javaue.properties")`

  これは、Java 用 Oracle GoldenGate フレームワークに使用されるデフォルトのプロパティ・ファイルを変更します。値は、クラスパスまたは通常のファイル・システム・パスにあるファイルへのパスです。

Java ハンドラの構成

Oracle GoldenGate Java API には、アクティブ・イベント・ハンドラの構成に使用するプロパティ・ファイルがあります。構成をテストするには、組込みファイル・ハンドラを使用できます。次に、プロパティ例とプロパティの説明 (# で始まるコメント行) を示します。

```
# the list of active handlers
gg.handlerlist=myhandler
# set properties on 'myhandler'
gg.handler.myhandler.type=file
gg.handler.myhandler.format=tx2xml.vm
gg.handler.myhandler.file=output.xml
```

このプロパティ・ファイルでは、次のように宣言しています。

- アクティブ・イベント・ハンドラ。この例では、`myhandler` という名前の 1 つのイベント・ハンドラがアクティブです。複数のハンドラをカンマで区切って指定できます。たとえば、`gg.handlerlist=myhandler, yourhandler` です。
- ハンドラの構成。この例では、myhandler は `file` タイプのハンドラと宣言されています (`gg.handler.myhandler.type=file`)。

  注意　設定可能な有効なプロパティのリストは、各タイプのハンドラ (JMS ハンドラ、ファイル・ライター・ハンドラなど) のドキュメントを参照してください。

- 出力の形式は、Velocity テンプレート `tx2xml.vm` によって定義されます。メッセージ形式を定義する独自のカスタム・テンプレートを指定できます。Java クラスパスを基準としてテンプレートのパスを指定します (これは後述します)。

このプロパティ・ファイルは、実際、取得されたトランザクションを出力ファイル output.xml に書き込む完全な例です。他のハンドラ・タイプは、キーワード **jms_text**(または **jms**)、**jms_map**、**singlefile**(ロールしないファイル) などを使用して指定できます。カスタム・ハンドラを実装できます。この場合、ハンドラの Java クラスの完全修飾名です。

第 7 章

UE: ユーザー・イグジットの実行

この章では、ユーザー・イグジットによって消費される証跡をプライマリ Extract ですでに生成してあるものとします。

アプリケーションの起動

ユーザー・イグジットを実行し、Java アプリケーションを実行するために必要なのは、既存の証跡ファイルとこれに対応するソース定義ファイルのみです。後述の例では、単純な TCUSTMER および TCUSTORD 証跡 (Oracle GoldenGate ソフトウェア・ダウンロードに含まれるデモ SQL に対応) および証跡に使用されるデータ型を定義するソース定義ファイルが使用されます。

注意 **ユーザー・イグジットは、実行にデータベースへのアクセスを必要としません。しかし、Extract プロセスは、証跡データを表すメタデータを必要とします。Extract はデータベースにログインしてメタデータを取得する必要があります。あるいは、ソース定義ファイルを指定することもできます。いずれの場合も、ユーザー・イグジットを使用する場合、Extract は PASSTHRU モードにできません。**

ユーザー・イグジットを実行するには、単に GGSCI から Extract プロセスを起動します。

```
GGSCI> START EXTRACT javaue
GGSCI> INFO EXTRACT javaue
```

INFO コマンドでは、次のような情報が返されます。

```
EXTRACT JAVAUE Last Started 2011-08-25 18:41 Status RUNNING
Checkpoint Lag 00:00:00 (updated 00:00:00 ago)
Log Read Checkpoint File ./dirdat/bb000000
2011-09-24 12:52:58.000000 RBA 2702
```

Extract プロセスが実行され、ファイル・ハンドラが使用される (前述の例のとおり) と、Oracle GoldenGate インストール・ディレクトリ (Extract 実行可能ファイルと同じディレクトリ) に出力ファイル output.xml が生成されます。

プロセスが起動しない場合や異常終了する場合、90 ページの「エラーのチェック」を参照してください。

証跡の先頭でのアプリケーションの再起動

ユーザー・イグジットを実行する Extract には、ユーザー・イグジット・チェックポイントと Extract チェックポイントの 2 つのチェックポイントがあります。Extract を再度実行する前に、両チェックポイントをリセットする必要があります。

1. ユーザー・イグジット・チェックポイント・ファイルを削除します。

サンプル・プロパティ・ファイルでは、ユーザー・イグジット・プロパティ・ファイルに goldengate.userexit.chkptprefix=JAVAUE_ が含まれています。

Windows: `cmd> del JAVAUE_javawriter.chkpt`

UNIX: `$ rm JAVAUE_javawriter.chkpt`

> **注意** 本番システムでチェックポイントを変更したり、ユーザー・イグジット・チェックポイント・ファイルを削除しないでください。

2. Extract を証跡データの先頭にリセットします。

```
GGSCI> ALTER EXTRACT JAVAUE, EXTSEQNO 0, EXTRBA 0
```

3. Extract を再起動します。

```
GGSCI> START JAVAUE
GGSCI> INFO JAVAUE

EXTRACT    JAVAUE    Last Started 2011-08-25 18:41   Status RUNNING
Checkpoint Lag       00:00:00 (updated 00:00:00 ago)
Log Read Checkpoint  File ./dirdat/ps000000
                     2011-09-24 12:52:58.000000  RBA 2702
```

Extract プロセスのステータスが実行中になるには、数秒かかる場合があります。レポート・ファイルをチェックして、異常終了したか、起動中のままかを確認します。

```
GGSCI> VIEW REPORT JAVAUE
```

第 8 章

UE: イベント・ハンドラの構成

イベント・ハンドラの指定

トランザクション、操作およびメタデータ・イベントの Java での処理は次のようになります。

- Oracle GoldenGate Extract がローカル証跡データを読み取り、トランザクション、操作およびデータベース・メタデータをユーザー・イグジットに渡します。メタデータは、ソース定義ファイルから、またはデータベースでの問合せによって取得します。
- Java フレームワークがイベントを発行します。イベントは、オプションでカスタム・イベント・フィルタによってフィルタされます。
- ハンドラ (イベント・リスナー) がこれらのイベントを処理し、トランザクション、操作およびメタデータを処理します。特定のタイプのターゲットにカスタム・フォーマッタが適用される場合があります。

既存のハンドラがいくつかあります。

- MapMessage を使用するか、TextMessage をカスタマイズ可能なフォーマッタと組み合せて使用し、JMS プロバイダに送信するメッセージ・ハンドラ。
- JMS メッセージを Oracle アドバンスト・キューイング (AQ) に送信する、専用のメッセージ・ハンドラ。
- 1 つのファイルまたはローリング・ファイルに書き込むファイル・ライター・ハンドラ。

> **注意** ファイル・ライター・ハンドラは JMS TextMessage ハンドラと完全に同じフォーマッタを使用できるため、ファイル・ライター・ハンドラは開発ユーティリティとして特に有用です。ファイル・ライターを使用すると、JMS に実際にメッセージを送らずに JMS 用のフォーマッタのテストとチューニングを簡単に行えます。

イベント・ハンドラはメイン Java プロパティ・ファイルを使用して構成できます。あるいは、オプションで別のプロパティ・ファイルから直接プロパティを読み込む場合もあります (ハンドラの実装によって異なります)。ハンドラのプロパティは次の構文を使用して設定されます。

```
gg.handler.{name}.someproperty=somevalue
```

これによって、プロパティ・ファイルで `{name}` で識別されるハンドラ・インスタンスのプロパティ *someproperty* が値 *somevalue* に設定されます。この `{name}` はプロパティ・ファイルで使用され、アクティブ・ハンドラを定義し、そのプロパティを設定します。これはユーザー定義です。

実装上の注意 (Java 開発者向け): 前述の例の後、ハンドラがインスタンス化されると、メソッド `void setSomeProperty(String value)` がハンドラ・インスタンスでコールされ、*somevalue* が渡されます。JavaBean PropertyEditor もハンドラに対して定義できます。この場合、文字列は、セッター・メソッドに対して適切な型に自動的に変換されます。たとえば、Java アプリケーション・プロパティ・ファイルで次のような設定だとします。

```
# the list of active handlers: only two are active
gg.handlerlist=one, two

# set properties on 'one'
gg.handler.one.type=file
gg.handler.one.format=com.mycompany.MyFormatter
gg.handler.one.file=output.xml

# properties for handler 'two'
gg.handler.two.type=jms_text
gg.handler.two.format=com.mycompany.MyFormatter
gg.handler.two.properties=jboss.properties
# set properties for handler 'foo'; this handler is ignored
gg.handler.foo.type=com.mycompany.MyHandler
gg.handler.foo.someproperty=somevalue
```

タイプによってハンドラ・クラスが識別されます。他のプロパティは、作成されるハンドラのタイプによって異なります。別個のプロパティ・ファイルがハンドラ (JMS ハンドラなど) の初期化に使用される場合、プロパティ・ファイルはクラスパスにあります。たとえば、プロパティ・ファイルが {*gg_install_dir*}/dirprm/foo.properties にある場合、プロパティ・ファイルで gg.handler.{*name*}.properties=foo.properties のように指定します。

JMS ハンドラ

メイン Java プロパティ・ファイルは、アクティブ・ハンドラを識別します。JMS ハンドラは、必要に応じて JMS 固有の構成用の個別のプロパティ・ファイルを使用できます。これによって、複数の JMS ハンドラが同時に稼働するよう構成できます。

いくつかの JMS プロバイダ (JBoss、TIBCO、Solace、ActiveMQ、WebLogic) 用にサンプルが含まれています。ご使用の環境用の出発点として特定の JMS プロバイダ用のプロパティ・ファイルを選択できます。JMS プロバイダごとに設定は多少異なります。環境に特有の設定もあります。

Java jar (ggjava) のインストール・ディレクトリには、コア・アプリケーション jar (ggjava.jar) とそのその依存性が resources/lib/*.jar に含まれています。リソース・ディレクトリはすべての依存性と構成を含み、クラスパス内にあります。

JMS クライアント jar がシステムにすでにある場合、直接参照し、コピーせずにクラスパスに追加できます。

指定可能な JMS ハンドラは 4 種類あります。

- **jms:** テキスト・メッセージをトピックまたはキューに送信します。メッセージは Velocity テンプレートを使用するか、Java でフォーマッタを記述してフォーマットします。ファイルへの書込みの際、同じフォーマッタを jms_text message に使用できます。(jms_text は jms と同義です。)
- **aq :** テキスト・メッセージを Oracle アドバンスト・キューイング (AQ) に送信します。aq ハンドラは、AQ への配信用に構成された jms ハンドラです。メッセージは Velocity テンプレートまたはカスタム・フォーマッタを使用してフォーマットできます。
- **jms_map:** JMS MapMessageをトピックまたはキューに送信します。メッセージのJMSTypeを表の名前に設定します。メッセージの本体は次のメタデータとそれに続く列名と列値のペアで構成されます。
  - GG_ID : この操作を一意に識別するレコードの位置
  - GG_OPTYPE: SQL のタイプ (挿入 / 更新 / 削除)

- GG_TABLE : 操作が行われた表の名前
- GG_TIMESTAMP: 操作のタイムスタンプ

ファイル・ハンドラ

ファイル・ハンドラは、実際のターゲットが JMS で、メッセージ形式がカスタム Java または Velocity テンプレートを使用して開発されている場合にメッセージ形式の確認に使用されることが多くあります。ファイル・ハンドラを使用するプロパティ・ファイルを次に示します。

```
# one file handler active, using velocity template formatting
gg.handlerlist=myfile
gg.handler.myfile.type=file
gg.handler.myfile.rollover.size=5M
gg.handler.myfile.format=sample2xml.vm
gg.handler.myfile.file=output.xml
```

この例では、1 つのハンドラを使用 (JMS ハンドラとファイル・ハンドラが同時に使用されることはある) して、output.xml という名前のファイルに書き込みます。sample2xml.vm という名前の Velocity テンプレートを使用します。テンプレートはクラスパスを介して特定されます。

カスタム・ハンドラ

カスタム・ハンドラのコーディングの詳細は、86 ページの「Java でのカスタム・ハンドラのコーディング」を参照してください。

出力のフォーマット

前述のとおり、既存の JMS およびファイル出力ハンドラはプロパティ・ファイルを使用して構成できます。各ハンドラには、設定可能な固有のプロパティがあります。たとえば、出力ファイルをファイル・ハンドラに設定したり、JMS 宛先を JMS ハンドラに設定できます。これらの両ハンドラでカスタム・フォーマッタも指定できます。同じフォーマッタを両方のハンドラに使用できます。カスタム・フォーマット用の Java コードを記述するかわりに、Velocity テンプレートを指定できます。詳細は、83 ページの「カスタム・フォーマット」を参照してください。

レポート

Extract プロセスが停止すると、スループットおよび処理されたデータの量に関するサマリー統計が生成されます。また、統計は、定期的に (指定した時間の経過後または指定したレコード数の処理後) 書き込まれます。時間とレコード数の両方が指定された場合、いずれかのイベントが発生すると、レポートが生成されます。これらの統計サマリーは、Oracle GoldenGate レポート・ファイルおよびユーザー・イグジット・ログ・ファイルに書き込まれます。

第 9 章

VAM: メッセージ配信プロパティ

この章では、次のもののためのプロパティ・ファイルの構成に使用できるオプションについて説明します。

- **ユーザー・イグジット・プロパティ**
- **Java アプリケーション・プロパティ**

プロパティ・ファイルは、Oracle GoldenGate インストールの場所の `dirprm` ディレクトリに配置します。ユーザー・イグジット・プロパティと Java アプリケーション・プロパティを含めたプロパティ・ファイルは、次の環境変数を使用して設定します。

```
SETENV (GGS_USEREXIT_CONF = "dirprm/javaue.properties")
```

オプションで、Java アプリケーション・プロパティとネイティブ・ユーザー・イグジット・ライブラリ・プロパティを別のプロパティ・ファイルに含めることができます。これを行うには、`GGS_USEREXIT_CONF` をユーザー・イグジット・プロパティ・ファイルに設定し、`GGS_JAVAUSEREXIT_CONF` を Java アプリケーション・プロパティ・ファイルに設定します。

プロパティ・ファイル内のプロパティにはすべて `fully.qualified.name=value` の形式が使用されます。値は、整数、ブール値、1 つの文字列またはカンマ区切りの文字列です。

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・アウトできます。ただし、行末にコメントを置くことはできません。行全体をコメントにするか、プロパティをコメントにします。

ユーザー・イグジット・プロパティ

次のプロパティでは、ログ・ファイルおよびロギングの特性を設定します。

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。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。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
```

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で実行されます。

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

次のプロパティは、ライター・タイプのすべてのユーザー・イグジットに適用され、ユーザー・イグジット固有ではありません。

goldengate.userexit.writers

ライターの名前を指定します。これは常に `javawriter` で、変更しないでください。

次に例を示します。

```
goldengate.userexit.writers=javawriter
```

ファイル内の他のすべてのプロパティにライター名 javawriter を接頭辞として付けます。

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。次に例を示します。

```
goldengate.userexit.chkptprefix=javaue_
```

goldengate.userexit.nochkpt

ユーザー・イグジット・チェックポイント・ファイルを無効または有効にします。デフォルトは `false` で、チェックポイント・ファイルは有効です。トランザクションがターゲットでサポートされ、有効な場合、このプロパティを `true` に設定します。

たとえば、JMS がターゲットで、JMS ローカル・トランザクションが有効 (デフォルト) な場合、`goldengate.userexit.nocheckpt=true` を設定し、ユーザー・イグジット・チェックポイント・ファイルを無効にします。ハンドラで `localTx=false` を設定し、JMS トランザクションが無効な場合、`goldengate.userexit.nochkpt=false` を設定してユーザー・イグジット・チェックポイント・ファイルを有効にします。

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

goldengate.userexit.usetargetcols

ターゲット列へのマッピングが可能かどうかを指定します。デフォルトは、`false` で、ターゲット・マッピングなしです。

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

JVM 起動オプション

次のオプションでは、Java Runtime Environment を構成します。特に、これによって JVM が使用される最大メモリーが指定されます。Java のメモリー不足エラーになる場合、これらの設定を編集します。

javawriter.bootoptions

ユーザー・イグジットで JVM を起動する場合に適用されるクラスパスおよび起動オプションを指定します。パスには、UNIX/Linux の場合コロン (:)、Windows の場合セミコロン (;) の区切り文字が必要です。ここでは、ヒープ・サイズ、クラスパスなど、JVM に対する様々なオプションを指定します。

-Xms: 初期 Java ヒープ・サイズ

-Xmx: 最大 Java ヒープ・サイズ

-**Djava.class.path:** メイン・アプリケーション jar である `ggjava.jar` を指定 (最低でもこれは指定) するクラスパス。JMS プロバイダ jar などのその他の jar もここで指定できます。また、これらを Java アプリケーション・プロパティ・ファイルで指定することもできます。

-verbose:jni: 詳細モードで実行します (JNI 用)

次に例を示します (すべて 1 行に記述します)。

```
javawriter.bootoptions= -Djava.class.path=ggjava/ggjava.jar
-Dlog4j.configuration=my-log4j.properties -Xmx512m
```

log4j.configuration プロパティには、log4j プロパティ・ファイルの完全修飾 URL を指定できます。デフォルトでは、このファイルはクラスパスで検索されます。独自の log4j 構成を使用することも、あらかじめ構成された log4j 設定である `log4j.properties`(デフォルト・レベルのロギング)、`debug_log4j.properties`(デバッグ・ロギング) または `trace_log4j.properties`(非常に詳細なロギング) のいずれかを使用することもできます。

統計およびレポート

ユーザー・イグジットを使用すると、Extract は、イグジットによって処理されるレコードは無視されるとみなします。これによって、標準の Oracle GoldenGate レポーティングは不完全になります。Java 用 Oracle GoldenGate では、独自のレポートを追加し、この問題に対応します。

統計は、`t` 秒ごとまたは `n` レコードごとに (両方が指定された場合、先に満たされた方の基準)、レポートされます。

ユーザー・イグジット共有ライブラリ (C 側) によって保持されるレコードと、Java ライブラリから取得されるレコードの 2 セットの統計レコードが記録されます。Java 側から受信されるレポートは、個々のハンドラによってフォーマットされ、返されます。

ユーザー・イグジット統計には、操作の総数、トランザクションとその速度が含まれます。

javawriter.stats.display

統計の Oracle GoldenGate レポート・ファイルおよびユーザー・イグジット・ログ・ファイルへの出力を制御します。

次の例では、これらの統計を出力します。

```
javawriter.stats.display=true
```

javawriter.stats.full

C 側からの統計に加え、Java 側からの統計の出力を制御します。

Java 側の統計はより詳細ですが、オーバーヘッドも増えます。このため、統計のレポートの頻度が高い場合、詳細度の低いサマリーで十分です。`stats.full` プロパティは `false` に設定することをお薦めします。

次の例では、C 以外に Java 統計を出力します。

```
javawriter.stats.full=true
```

javawriter.stats.{time, numrecs}

統計がレポートされる間隔 (秒) またはレコード数を指定します。デフォルトでは、毎時または 10000 レコードごと (いずれか先に起きた方) にレポートします。

たとえば、10 分ごとまたは 1000 レコードごとにレポートするには、次のように指定します。

```
javawriter.stats.time=600
javawriter.stats.numrecs=1000
```

Java アプリケーション統計は、ハンドラによって異なります。

- すべてのハンドラについて、少なくとも総経過時間、処理時間、操作数、トランザクション数があります。
- JMS ハンドラの場合、送受信されたバイト数合計もあります。
- レポートはテンプレートを使用してカスタマイズできます。

Java アプリケーション・プロパティ

Java アプリケーション・プロパティ・ファイルで設定できるプロパティを次に定義します。

すべてのハンドラ用のプロパティ

次のプロパティがすべてのハンドラに適用されます。

gg.handlerlist

ハンドラ・リストは、アクティブ・ハンドラのカンマ区切りのリストです。これらの値はプロパティ・ファイルの以降の部分で使用され、各ハンドラが構成されます。次に例を示します。

```
gg.handlerlist=name1, name2
gg.handler.name1.propertyA=value1
gg.handler.name1.propertyB=value2
gg.handler.name1.propertyC=value3
gg.handler.name2.propertyA=value1
gg.handler.name2.propertyB=value2
gg.handler.name2.propertyC=value3
```

`handlerlist` プロパティを使用すると、完全に構成されたハンドラをプロパティ・ファイルに含め、ハンドラリストから削除することで無効にすることができます。

gg.handler.{name}.type

ハンドラのタイプは、組込みハンドラ用にあらかじめ定義された値または完全修飾 Java クラス名です。構文は次のとおりです。

```
gg.handler.{name}.type=jms|jms_map|aq|singlefile|rolling| {com.foo.MyHandler}
```

条件 : 最後のハンドラ以外すべて、事前定義のハンドラです。

jms: トランザクション、操作およびメタデータをフォーマットされたメッセージとして JMS プロバイダに送信します。

aq: トランザクション、操作およびメタデータをフォーマットされたメッセージとして Oracle アドバンスト・キューイング (AQ) に送信します。

jms_map: JMS マップ・メッセージを送信します。

singlefile: ディスク上の 1 つのファイルに書き込みますが、ファイルをロールしません。

rolling: トランザクション、操作およびメタデータをディスク上のファイルに書き込み、特定のサイズまたは特定の時間を超えると、ファイルをロールオーバーします。

カスタム Java クラス : Java 用 Oracle GoldenGate の AbstractHandler クラスを拡張するクラスは、トランザクション、操作、メタデータ・イベントを処理できます。

フォーマットされた出力用のプロパティ

次のプロパティは、フォーマットされた出力を生成できるすべてのハンドラに適用されます。これには、次のようなものがあります。

- `jms_text` **ハンドラ** (`jms_map` **ハンドラ**ではない)
- `aq` ハンドラ
- フォーマットされた出力をファイルに書き込む `singlefile` および `rolling` ハンドラ

gg.handler.{name}.format

操作およびトランザクションを JMS またはファイルに送信するメッセージに変換するために使用される形式を指定します。形式はハンドラごとに一意に指定されます。値は次のとおりです。

- **Velocity テンプレート**
- **Java クラス名** (完全修飾。指定されるクラスはフォーマッタのタイプである必要があります)
- 区切られた値の場合の **csv**(カンマ区切りなど。デリミタはカスタマイズできます)
- 固定長フィールドの場合の **fixed**
- 次のような**組込みフォーマッタ**
  - `xml`: デモ XML 形式 (この形式は今後のリリースで変更される可能性があります)
  - `xml2`: 内部 XML 形式 (この形式は今後のリリースで変更される可能性があります)

たとえば、カスタム Java クラスを指定するには、次のようにします。

```
gg.handlerlist=abc
gg.handler.abc.format=com.mycompany.MyFormat
```

Velocity テンプレートの場合、次のようにします。

```
gg.handlerlist=xyz
gg.handler.xyz.format=path/to/sample.vm
```

テンプレートを使用する場合、クラスパス内のディレクトリまたは jar を基準にファイルが検索されます。デフォルトでは、Oracle GoldenGate インストール・ディレクトリがクラスパス内にあるため、前述のテンプレートは、Oracle GoldenGate のインストール場所の `dirprm` ディレクトリに配置できます。

デフォルトの形式は、組込み XML フォーマッタを使用することです。

gg.handler.{name}.includeTables

このハンドラによって含められる表のリストを指定します。表のスキーマ (または所有者) が指定される場合、そのスキーマのみが表名に一致します。それ以外の場合、表名は任意のスキーマに一致します。表のリストはカンマ区切りで指定されます。

たとえば、ハンドラに表 foo.customer および bar.orders のみを処理させるには、次のようにします。

```
gg.handler.myhandler.includeTables=foo.customer, bar.orders
```

注意 **表単位で操作を選択的に処理するには、ハンドラは操作モードで処理している必要があります。ハンドラがトランザクション・モードで処理しており、1 つのトランザクションに複数の表にまたがる複数の操作が含まれている場合、いずれかの表が表のリストに一致すれば、トランザクションは含められます。**

gg.handler.{name}.excludeTables

このハンドラによって除外される表のリストを指定します。表のスキーマ (または所有者) が指定される場合、そのスキーマのみが表名に一致します。それ以外の場合、表名は任意のスキーマに一致します。表のリストはカンマ区切りで指定されます。たとえば、すべてのスキーマの `date_modified` *以外*のすべての表に対するすべての操作をハンドラが処理するには、次のようにします。

```
gg.handler.myhandler.excludeTables=date_modified
```

gg.handler.{name}.mode、*gg.handler .{name}.format.mode*

1 メッセージ当たり 1 つの操作を出力する (`op`) か、1 メッセージ当たり 1 つのトランザクションを出力する (`tx`) か指定します。デフォルトは、`op` です。カスタム・フォーマッタの場合、`format.mode` を使用します。

CSV および固定形式の出力用プロパティ

ハンドラが CSV または固定形式の出力を使用するよう設定されている場合、次のプロパティも設定されます。プロパティの多くは両方の形式に適用されますが、プロパティ設定に一意の接頭辞はありません。一意の設定が必要なハンドラが複数ある場合、これらのプロパティは個別のプロパティ・ファイルで設定できます。たとえば、2 つの JMS ハンドラがあり、それぞれ CSV または固定形式を使用する場合、次のようになります。

```
gg.handler.my_jms_handler1.type=jms_text
gg.handler.my_jms_handler1.format=csv
gg.handler.my_jms_handler1.properties=my-csv.properties
.
.
.
gg.handler.my_jms_handler2.type=jms_text
gg.handler.my_jms_handler2.format=fixed
gg.handler.my_jms_handler2.properties=my-fixed.properties
.
.
.
```

delim

フィールド間に使用するデリミタを指定します (デリミタを使用しない場合、値を設定しません)。次に例を示します。

```
delim=,
```

quote

列値が引用府付きの場合、使用する引用文字を指定します。次に例を示します。

```
quote='
```

metacols

レコードの先頭、すべての列データの前に出力するメタデータ列値を指定します。次のものを出力順に指定します。

- **position**: 証跡内のレコードの一意の位置インジケータ
- **opcode**: レコードの挿入、更新または削除に対して `I`、`U` または `D` (`insertChar`、`updateChar`、`deleteChar` を参照)
- **txind**: `0`= 始まり、`1`= 中間、`2`= 終わり、`3`= トランザクション全体などのトランザクション・インジケータ (`beginTxChar`、`middleTxChar`、`endTxChar`、`wholeTxChar` を参照)
- **opcount**: 0 から始まるトランザクション内のレコードの位置
- **schema**: レコードの表のスキーマ / 所有者
- **tableonly**: 表のみ (スキーマ / 所有者なし)
- **table**: 表の完全名 (schema.table)
- **timestamp**: レコードのコミット・タイムスタンプ

次に例を示します。

```
metacols=opcode, table, txind, position
```

missingColumnChar、*presentColumnChar*、*nullColumnChar*

次の列値に対する特別な列接頭辞を指定します。

- **存在** : 列値は証跡にあり、NULL ではありません。
- **欠落** : 列値は証跡にありません。値があるか NULL かは不明です。ソース・データベース・トランザクション・ログから取得されませんでした。
- **null**: 列値は NULL に設定されます。

これらの特別な状態を表すために使用される文字はカスタマイズできます。デフォルトでは、空の文字列に設定され、出現しません。次に例を示します。

```
missingColumnChar=M
presentColumnChar=P
nullColumnChar=N
```

beginTxChar、*middleTxChar*、*endTxChar*、*wholeTxChar*

レコードをトランザクションの始まり、中間または終わりと識別するために使用されるヘッダー・メタデータ文字 (`metacols` を参照) を指定します。1 つの操作が完全なトランザクションで構成される場合、トランザクション全体です。次に例を示します。

```
beginTxChar=B
middleTxChar=M
endTxChar=E
wholeTxChar=W
```

insertChar、updateChar、deleteChar

挿入、更新および削除を識別する文字を指定します。デフォルトでは、これらは I、U および D です。たとえば、挿入、更新および削除操作に対して I、U および D のかわりに、INS、UPD および DEL を使用するには、次のようにします。

```
insertChar=INS
updateChar=UPD
deleteChar=DEL
```

endOfLine

行末文字を指定します。

- ネイティブ・プラットフォーム : EOL
- ニュートラル (UNIX スタイル \n): CR
- Windows (\r\n): CRLF

次に例を示します。

```
endOfLine=CR
```

justify

固定フィールドを右詰めにするか、左詰めにするかを指定します。、次に例を示します。

```
justify=left
```

includeBefores

ビフォア・イメージを出力に含めるかどうかを制御します。証跡にビフォア・イメージがある必要があります。次に例を示します。

```
includeBefores=false
```

ファイル・ライター・プロパティ

次のプロパティは、出力をファイルに書き込むハンドラ (ファイル・ハンドラおよび単一ファイル・ハンドラ) にのみ適用されます。

gg.handler.{name}.file

指定されたハンドラの出力ファイルの名前を指定します。ハンドラがローリング・ファイルの場合、この名前は、ロールされたファイルの名前の導出に使用されます。デフォルトのファイル名は、output.xml です。

gg.handler.{name}.append

ファイルが追加される (true) か、再起動時に上書きされる (false) かを制御します。

gg.handler.{name}.rolloverSize

ファイル・ハンドラを使用する場合、ロールオーバーが試行されるファイルのサイズを指定します。ファイル・サイズは最低このサイズですが、ほとんどの場合、これより大きいです。操作およびトランザクションはファイル間で分割されません。サイズはバイト数で指定されますが、接尾辞を指定して MB または KB を識別できます。次に例を示します。

```
gg.handler.myfile.rolloverSize=5M
```

デフォルトのロールオーバー・サイズは、10MB です。

JMS ハンドラ・プロパティ

次のプロパティが JMS ハンドラに適用されます。これらの値のいくつかは、ハンドラの名前を使用して Java アプリケーション・プロパティ・ファイルで定義できます。他のプロパティは、個別のプロパティ・ファイルに含めることができます。これは、一度に複数の JMS ハンドラを使用する場合、有用です。次に例を示します。

```
gg.handler.myjms.type=jms_text
gg.handler.myjms.format=xml
gg.handler.myjms.properties=weblogic.properties
```

Velocity テンプレートとフォーマット・プロパティ・ファイル同様、この追加 JMS プロパティ・ファイルはクラスパスで検索されます。dirprm ディレクトリはデフォルトでクラスパスに含まれているため、前述のプロパティ・ファイル weblogic.properties は、{*gg_install_dir*}/dirprm/weblogic.properties にあります。

Java アプリケーション・プロパティ・ファイルの設定は、追加の JMS プロパティ・ファイル (前述の例では weblogic.properties) で設定された対応する値をオーバーライドします。次の例では、宛先プロパティが Java アプリケーション・プロパティ・ファイルで指定されています。これは、2 つのハンドラ myjms1 および myjms2 に同じデフォルト接続情報を使用しますが、ターゲット宛先キューをカスタマイズします。

```
gg.handler.myjms1.type=jms_text
gg.handler.myjms1.destination=queue.sampleA
gg.handler.myjms1.format=sample.vm
gg.handler.myjms1.properties=tibco-default.properties
gg.handler.myjms2.type=jms_map
gg.handler.myjms2.destination=queue.sampleB
gg.handler.myjms2.properties=tibco-default.properties
```

プロパティを設定するには、次のようにハンドラ名を接頭辞として指定します。

```
gg.handlerlist=sample,sample2
gg.handler.sample.type=jms_text
gg.handler.sample.format=my_template.vm
gg.handler.sample.destination=gg.myqueue
gg.handler.sample.queueortopic=queue
gg.handler.sample.connectionUrl=tcp://host:61616?jms.useAsyncSend=true
gg.handler.sample.useJndi=false
gg.handler.sample.connectionFactory=ConnectionFactory
gg.handler.sample.connectionFactoryClass=\
    org.apache.activemq.ActiveMQConnectionFactory
gg.handler.sample.connection.Url=
tcp://localhost:61616?jms.useAsyncSend=true
gg.handler.sample.timeToLive=50000
```

標準 JMS 設定

次に、設定可能な JMS プロパティと許容される値について簡単に説明します。これらは、jms_text (TextMessage) および jms_map (MapMessage) の両方の JMS ハンドラ・タイプに適用されます。

gg.handler.{name}.destination

メッセージが送信されるキューまたはトピック。これは、JMS サーバーで適切に構成される必要があります。標準的な値は、`queue/A`、`queue.Test`、`example.MyTopic` などです。

gg.handler.{name}.user

JMS サーバーへのメッセージの送信に必要なユーザー名 (オプション)。

gg.handler.{name}.password

JMS サーバーへのメッセージの送信に必要なパスワード (オプション)。

gg.handler.{name}.queueOrTopic

ハンドラがキューに送信する (単一受信者) か、トピックに送信する (パブリッシュ / サブスクライブ) か。これは、JMS プロバイダで適切に構成される必要があります。構文は次のとおりです。

```
gg.handler.{name}.queueOrTopic=queue|topic
```

条件 : **queue**: メッセージは読み取られると、削除されます。

topic: メッセージはパブリッシュされ、複数のサブスクライバに配信されます。

gg.handler.{name}.persistent

配信モードが永続に設定されているかどうか。メッセージが永続の場合、クライアントの送信操作の一環としてメッセージを安定的なストレージに記録するよう JMS プロバイダを構成する必要があります。構文は次のとおりです。

```
gg.handler.{name}.persistent=true|false
```

gg.handler.{name}.priority

JMS では、0 を最低、9 を最高とする 10 段階の優先度の値が定義されます。クライアントは 04 を通常優先度のグラデーション、59 を優先優先度のグラデーションとしてみなします。デフォルトでは優先度は 4 に設定されます。

gg.handler.{name}.timeToLive

生成されたメッセージがメッセージ・システムによって保持される時間のデフォルトの長さ。ディスパッチ時間からのミリ秒。存続時間はデフォルトでは 0 に設定されます (0 は無制限)。

gg.handler.{name}.connectionFactory

JNDI を介して検索する接続ファクトリの名前。

gg.handler.{name}.useJndi

`usejndi` が `false` の場合、JNDI は JMS クライアントの構成に使用されません。かわりに、ファクトリおよび接続が明示的に構築されます。構文は次のとおりです。

```
gg.handler.{name}.useJndi=true|false
```

gg.handler.{name}.connectionUrl

usejndi=false の場合にのみ接続の明示的な作成に使用される接続 URL。

gg.handler.{name}.connection.FactoryClass

`usejndi=false` の場合にのみ使用される ConnectionFactoryClass。JNDI を介してファクトリにアクセスしない場合、このプロパティの値は、ファクトリ・オブジェクトを明示的にインスタンス化および構築する Java クラス名です。

gg.handler.{name}.localTX

`false` に設定される場合、ローカル・トランザクションは使用されません。

gg.handlerlist.nop

他の JMS プロパティに加え、JMS メッセージの送信を完全に無効にするようグローバルに設定できるデバッグ "nop" プロパティがあります。これは、テスト目的でのみ使用されます。従前どおりイベントが生成されて処理され、メッセージが構築されます。これは、メッセージ生成のパフォーマンスのテストに使用されます。`true` または `false` に設定できます (デフォルトは、`false` です)。次に例を示します。

```
gg.handlerlist.nop=true
```

JNDI プロパティ

これらの JNDI プロパティは、接続ファクトリと宛先を検索するための初期コンテキストへの接続に必要です。

```
java.naming.provider.url={url}
java.naming.factory.initial={java-class-name}
```

JNDI セキュリティが有効な場合、次のプロパティを設定できます。

```
java.naming.security.principal={user-name}
java.naming.security.credentials={password-or-other-authenticator}
```

次に例を示します。

```
java.naming.provider.url= t3://localhost:7001
java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory
java.naming.security.principal=jndiuser
java.naming.security.credentials=jndipw
```

一般プロパティ

次のプロパティは、ユーザー・イグジット Java フレームワーク用に使用される一般プロパティです。

gg.classpath

クラスパスに追加するディレクトリまたは jar を指定します。

gg.report.format

レポート形式のカスタマイズに使用するテンプレートを指定します。

第 10 章

UE: カスタム・フィルタ、フォーマッタおよびハンドラの開発

イベント・フィルタ、組込みハンドラ用のカスタム・フォーマッタやカスタム・イベント・ハンドラを実装する Java コードを記述できます。Velocity テンプレートを介してカスタム形式を指定することもできます。

イベントのフィルタ

デフォルトでは、すべてのトランザクション、操作およびメタデータ・イベントが `DataSourceListener` イベント・ハンドラに渡されます。イベント・フィルタを実装して、ハンドラに送信するイベントをフィルタできます。たとえば、フィルタは、特定の列値を含む特定の表に対する特定の操作を選択します。

フィルタは加法的です。複数のフィルタがハンドラに設定されている場合、イベントがハンドラに渡されるには、すべてのフィルタが true を返す必要があります。

フィルタは、Java アプリケーション・プロパティ・ファイルを使用して構成できます。

```
# handler "foo" only receives certain events
gg.handler.one.type=jms
gg.handler.one.format=mytemplate.vm
gg.handler.one.filter=com.mycompany.MyFilter
```

フィルタをアクティブにするには、フィルタを記述し、ハンドラに設定します。追加のロジックを特定のハンドラに追加する必要はありません。

カスタム・フォーマット

次のようにして、組込みハンドラの出力フォーマットをカスタマイズできます。

- Java でのカスタム・フォーマッタの記述。または
- Velocity テンプレートの使用

Java でのカスタム・フォーマッタのコーディング

これより前の例で、同じフォーマッタ (`com.mycompany.MyFormatter`) を使用する JMS ハンドラと

ファイル出力ハンドラを示しています。次の例は、フォーマッタの実装方法の例です。

```
package com.mycompany.MyFormatter;

import com.goldengate.atg.datasource.DsOperation;
import com.goldengate.atg.datasource.DsTransaction;
import com.goldengate.atg.datasource.format.DsFormatterAdapter;
import com.goldengate.atg.datasource.meta.ColumnMetaData;
import com.goldengate.atg.datasource.meta.DsMetaData;
import com.goldengate.atg.datasource.meta.TableMetaData;
import java.io.PrintWriter;

public class MyFormatter extends DsFormatterAdapter {
    public MyFormatter() { }

    @Override
    public void formatTx(DsTransaction tx,
                DsMetaData meta,
                PrintWriter out)
    {
        out.print("Transaction: " );
        out.print("numOps=\'" + tx.getSize() + "\' " );
        out.println("ts=\'" + tx.getStartTxTimeAsString() + "\'");

        for(DsOperation op: tx.getOperations()) {
            TableName currTable = op.getTableName();
            TableMetaData tMeta = dbMeta.getTableMetaData(currTable);
            String opType = op.getOperationType().toString();
            String table = tMeta.getTableName().getFullName();
            out.println(opType + " on table \"" + table + "\":" );
            int colNum = 0;
            for(DsColumn col: op.getColumns())
            {
                ColumnMetaData cMeta = tMeta.getColumnMetaData( colNum++ );
                out.println(
                cMeta.getColumnName() + " = " + col.getAfterValue() );
            }
    }

    @Override
    public void formatOp(DsTransaction tx,
                DsOperation op,
                TableMetaData tMeta,
                PrintWriter out)
    {
        // not used...
    }
}
```

フォーマッタは、トランザクション全体のフォーマット (コミット後)、または各操作のフォーマット (受信後、コミット前) の方法を定義します。フォーマッタが操作モードの場合、`formatOp`(...) がコールされます。そうではない場合、トランザクション・コミット時に `formatTx`(...) がコールされます。

このカスタム・フォーマッタをコンパイルして使用するには、Java 用 Oracle GoldenGate jar をクラスパスに含め、コンパイルした .class ファイルを {gg_install_dir}/dirprm に配置します。

```
javac -d {gg_install_dir}/dirprm
-classpath ggjava/ggjava.jar MyFormatter.java
```

結果のクラス・ファイルは、resources/classes に (正しいパッケージ構造で) 配置されます。

```
{gg_install_dir}/dirprm/com/mycompany/MyFormatter.class
```

あるいは、カスタム・クラスを jar に含めることもできます。この場合、jar ファイルをユーザー・イグジット・プロパティを介して JVM クラスパスに含める (javawriter.bootoptions プロパティで java.class.path を使用) か、Java アプリケーション・プロパティ・ファイルを設定してカスタム jar を含めます。

```
# set properties on 'one'
gg.handler.one.type=file
gg.handler.one.format=com.mycompany.MyFormatter
gg.handler.one.file=output.xml
gg.classpath=/path/to/my.jar,/path/to/directory/of/jars/*
```

Velocity テンプレートの使用

Velocity テンプレートは、カスタム・フォーマット用の Java コードを記述するかわりの方法として、フォーマッタのプロトタイプを簡単に作成するよい代替方法です。たとえば、次のテンプレートを JMS またはファイル・ハンドラの形式として指定します。

```
Transaction: numOps='$tx.size' ts='$tx.timestamp'
#for each( $op in $tx )
operation: $op.sqlType, on table "$op.tableName":
#for each( $col in $op )
$op.tableName, $col.meta.columnName = $col.value
#end
#end
```

テンプレートの名前が sample.vm とすると、次のようにクラスパスに配置します。

```
{gg_install_dir}/dirprm/sample.vm
```

> **注意** Velocity テンプレートを使用する場合、ファイル名は接尾辞 .vm で終わる必要があります。そうでない場合、フォーマッタは Java クラスとみなされます。

テンプレートを使用するよう Java アプリケーション・プロパティ・ファイルを更新します。

```
# set properties on 'one'
gg.handler.one.type=file
gg.handler.one.format=sample.vm
gg.handler.one.file=output.xml
```

テンプレートを変更する場合、Java ソースを再コンパイルする必要はありません。テンプレートを保存して Java アプリケーションを再実行するのみです。アプリケーションを実行すると、次の出力が生成されます (表の名前は SCHEMA.SOMETABLE で、列は TESTCOLA および TESTCOLB とします)。

```
Transaction: numOps='3' ts='2008-12-31 12:34:56.000'
operation: UPDATE, on table "SCHEMA.SOMETABLE":
SCHEMA.SOMETABLE, TESTCOLA = value 123
SCHEMA.SOMETABLE, TESTCOLB = value abc
operation: UPDATE, on table "SCHEMA.SOMETABLE":
SCHEMA.SOMETABLE, TESTCOLA = value 456
SCHEMA.SOMETABLE, TESTCOLB = value def
operation: UPDATE, on table "SCHEMA.SOMETABLE":
SCHEMA.SOMETABLE, TESTCOLA = value 789
SCHEMA.SOMETABLE, TESTCOLB = value ghi
```

Java でのカスタム・ハンドラのコーディング

カスタム・ハンドラは、AbstractHandler を拡張して実装できます。

```
import com.goldengate.atg.datasource.*;
import static com.goldengate.atg.datasource.GGDataSource.Status;

public class SampleHandler extends AbstractHandler {

    @Override
    public void init(DsConfiguration conf, DsMetaData metaData) {
        super.init(conf, metaData);
        // ... do additional config...
    }

    @Override
    public Status operationAdded(DsEvent e, DsTransaction tx, DsOperation op) {
    ... }

    @Override
    public Status transactionCommit(DsEvent e, DsTransaction tx) { ... }

    @Override
    public Status metaDataChanged(DsEvent e, DsMetaData meta) { .... }

    @Override
    public void destroy() { /* ... do cleanup ... */ }

    @Override
    public String reportStatus() { return "status report..."; }
}
```

トランザクションが Extract から処理される際、ハンドラへのコール順序は次のようになります。

1. 初期化 :
   - まず、ハンドラが構築されます。
   - 次に、インスタンスでプロパティ・ファイルの値を使用してすべてのセッターがコールされます。
   - 最後に、ハンドラが初期化されます。トランザクションを受信する前に `init(...)` メソッドがコールされます。`init(...)` メソッドで `super.init(...)` をコールして基底クラスを適切に初期化することが重要です。

2. メタデータが受信されます。ユーザー・イグジットが、この実行時にまだ出現していない表での操作を処理する場合、メタデータ・イベントが発行され、`metadataChanged(...)` メソッドがコールされます。通常は、このメソッドを実装する必要はありません。`DsMetaData` は、新規データ・ソース・メタデータが受信されると自動的に更新されます。
3. トランザクションが開始されます。トランザクション・イベントが発行され、ハンドラで `transactionBegin(...)` メソッドが起動されます (記載していません)。この時点ではトランザクションに操作がないため、これは通常使用されません。
4. 操作が順次トランザクションに追加されます。これによって、各操作の追加のたびにハンドラで `operationAdded(...)` メソッドがコールされます。データ・ソース・メタデータ (この時点までに出現したすべての表メタデータを含む) とともに、これを含むトランザクションもメソッドに渡されます。トランザクションはまだコミットされておらず、コミットが受信される前に異常終了される可能性があることに注意してください。

   各操作にはトランザクションの列値が含まれます (Extract が圧縮更新を処理する場合、変更された値のみの可能性があります)。列値には、ビフォア値とアフター値の両方が含まれることがあります。
5. トランザクションがコミットされます。これによって、`transactionCommit(...)` メソッドがコールされます。
6. 定期的に `reportStatus` がコールされます。プロセスの停止時にもコールされます。通常、これによって処理の統計が表示されます (操作数 / 処理されたトランザクション、など)。

単純なプリンタ・ハンドラの完全な例を次に示します。トランザクション、操作およびメタデータの非常に基本的な情報をプリントアウトするのみです。ハンドラには、出力ファイル名を設定するためのプロパティ `myoutput` もあります。これは、Java アプリケーション・プロパティ・ファイルで次のように設定できます。

```
gg.handlerlist=sample
# set properties on 'sample'
gg.handler.sample.type=sample.SampleHandler
gg.handler.sample.myoutput=out.txt
```

カスタム・ハンドラを使用するには、次のようにします。

1. クラスをコンパイルします。
2. クラスをアプリケーション・クラスパスに含めます。
3. Java アプリケーション・プロパティ・ファイルのアクティブ・ハンドラのリストにハンドラを追加します。

ハンドラをコンパイルするには、Java 用 Oracle GoldenGate jar をクラスパスに含め、コンパイルした .class ファイルを `{gg_install_dir}/javaue/resources/classes` に配置します。

```
javac -d {gg_install_dir}/dirprm
-classpath ggjava/ggjava.jar SampleHandler.java
```

結果のクラス・ファイルは、次のように `resources/classes` に (正しいパッケージ構造で) 配置されます。

```
{gg_install_dir}/dirprm/sample/SampleHandler.class
```

注意 サンプル "hello world" 以外の Java アプリケーションの開発では、Ant または Maven を使用してアプリケーションをコンパイル、テストおよびパッケージ化します。javac を示した例は、例示目的のみです。

あるいは、カスタム・クラスを jar に含め、クラスパスに含めることができます。カスタム jar ファイルをユーザー・イグジット・プロパティを介して JVM クラスパスに含める (`javawriter.bootoptions` プロパティで `java.class.path` を使用) か、Java アプリケーション・プロパティ・ファイルを設定してカスタム jar を含めます。

```
# set properties on 'one'
gg.handler.one.type=sample.SampleHandler
gg.handler.one.myoutput=out.txt
gg.classpath=/path/to/my.jar,/path/to/directory/of/jars/*
```

任意のハンドラで、追加の個別 jar、ディレクトリ (リソースまたは jar になっていないクラス・ファイルを含む) または jar のディレクトリ全体を含めるようクラスパス・プロパティを設定できます。jar のディレクトリ全体を含めるには、Java 6 形式の構文を使用します。

```
c:/path/to/directory/* (Unix の場合： /path/to/directory/* )
```

ワイルドカード * のみ指定できます。ファイル・パターンは使用できません。これは、`.jar` 接尾辞で終わるディレクトリ内のすべてのファイルに自動的に一致します。複数の jar または複数のディレクトリを含めるには、システム固有のパス区切り文字 (UNIX ではコロン、Windows ではセミコロン) を使用するか、前述のようにプラットフォームに依存しないカンマを使用します。

ハンドラに多数のプロパティを設定する必要がある場合、パラメータ・ファイルにプロパティを含めるだけで、ハンドラの対応するセッターがコールされます。次に例を示します。

```
gg.handler.one.type=com.mycompany.MyHandler
gg.handler.one.myOutput=out.txt
gg.handler.one.myCustomProperty=12345
```

前述の例では、カスタム・ハンドラ内の次のメソッドが起動されます。

```
public void setMyOutput(String s) {
    // use the string...
} public void setMyCustomProperty(int j) {
    // use the int...
}
```

int、long、String、boolean などの標準の Java 型を使用できます。カスタム型の場合、カスタム・プロパティ・エディタを作成して String をカスタム型に変換できます。

追加リソース

Java API には Javadoc が用意されています。Javadoc は、カスタマイズおよび拡張に有用なインタフェースおよびクラスのみを配布するために、コア・パッケージ、クラスおよびインタフェースのセットに意図的に縮小されています。

各パッケージでは、わかりやすくするために一部のクラスが意図的に省略されています。重要なクラスは次のとおりです。

- `com.goldengate.atg.datasource.DsTransaction`: データベース・トランザクションを表します。トランザクションには、0 個以上の操作が含まれます。
- `com.goldengate.atg.datasource.DsOperation`: データベース操作 (挿入、更新、削除) を表します。操作には、データ変更イベントを表す 0 個以上の列値が含まれます。列索引は、Java API で 0 でオフセットされます。

- `com.goldengate.atg.datasource.DsColumn`: 列値を表します。列値は、ビフォア値とアフター値のコンポジットです。列値は存在する (値があるか null) 場合も欠落している (ソース証跡に含まれていない) 場合もあります。
  - `com.goldengate.atg.datasource.DsColumnComposite` は、コンポジットです。
  - `com.goldengate.atg.datasource.DsColumnBeforeValue` は、操作前の列値です (これはオプションで、操作に含まれていない場合があります)。
  - `com.goldengate.atg.datasource.DsColumnAfterValue` は、操作後の値です。
- `com.goldengate.atg.datasource.meta.DsMetaData`: 出現するすべてのデータベース・メタデータを表します。オブジェクトは最初空です。DsMetaData には、TableName をキーとして使用する TableMetaData の 0 個以上のインスタンスのハッシュ・マップが含まれます。
- `com.goldengate.atg.datasource.meta.TableMetaData`: 1 つの表のすべてのメタデータを表します。0 個以上の `ColumnMetaData` を含みます。
- `com.goldengate.atg.datasource.meta.ColumnMetaData`: データベースまたは Oracle GoldenGate ソース定義ファイルに定義されている列名およびデータ型を含みます。

詳細は、Javadoc を参照してください。

第 11 章
トラブルシューティング

この章にリストされたエラー・チェックを実行します。問題が特定できない場合、Oracle サポートにご連絡ください。

エラーのチェック

Java 用 Oracle GoldenGate の操作時発生する可能性のあるエラーには 2 つのタイプがあります。

- ユーザー・イグジットまたは VAM を実行する Extract プロセスが起動しないか、異常終了します。
- 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。

Extract プロセスが起動しないか異常終了する場合、処理の最初から最後まで順にエラー・メッセージをチェックします。

1. Oracle GoldenGate イベント・ログでエラーをチェックし、Extract レポート・ファイルを表示します。

   ```
   GGSCI> VIEW GGSEVT
   GGSCI> VIEW REPORT {extract name}
   ```

2. アプリケーション・ログ・ファイルをチェックします。

   ユーザー・イグジットの場合 :

   - ユーザー・イグジット・ライブラリのログ・ファイルにレポートされている最後のメッセージを確認します。ファイル名は、プロパティ・ファイルで設定されたログ・ファイル接頭辞 (`log.logname`) と現在の日付です。

     ```
     shell> more {log.logname}_{yyyymmdd}.log
     ```

   注意 : これは、Java アプリケーションのログ・ファイルではなく、共有ライブラリのみのログ・ファイルです。

3. ユーザー・イグジットまたは VAM が Java ランタイムを起動できなかった場合、log4j ログ・ファイルが存在します。

   ログ・ファイルの名前は log4j.properties ファイルに定義されています。デフォルトでは、ログ・ファイル名は `ggjava-{`*`version`*`}-log4j.log` で、*`version`* は使用される jar ファイルのバージョン番号です。次に例を示します。

   ```
   shell> more ggjava-*log4j.log
   ```

   Java アプリケーションのロギングの詳細レベルを設定するには、次のいずれかを行います。

   - より詳細なレベルで記録するよう現在の log4j プロパティを編集します。

- プロパティ・ファイルを編集して既存の log4j 構成の 1 つを再利用します。

  ```
  {javawriter or jvm}.bootoptions=-Djava.class.path=ggjava/ggjava.jar
  -Dlog4j.configuration=debug-log4j.properties Xmx512m
  ```

あらかじめ構成されたこれらの log4j プロパティ・ファイルはクラスパスにあり、次の場所にインストールされます。

```
./ggjava/resources/classes/*log4j.properties
```

4. これらのログ・ファイルで問題の原因が明らかにならない場合、Extract プロセスをシェルから直接 (GGSCI 外で) 実行し、stderr および stdout をより簡単に監視でき、環境変数を確認できるようにします。次に例を示します。

   ```
   shell> EXTRACT PARAMFILE dirprm/javaue.prm
   ```

プロセスは正常に実行されるが、データが正しくないか、存在しない場合、ユーザー・イグジット用に記述したカスタム・フィルタ、フォーマッタまたはハンドラでエラーがないかチェックします。

ユーザー・イグジット Extract を証跡の先頭から再起動するには、66 ページを参照してください。

コア Oracle GoldenGate ソフトウェアのトラブルシューティングの詳細は、『*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。

異常終了後のリカバリ

Extract パラメータ RECOVERYOPTIONS は、リリース 10 以降の証跡では APPENDMODE がデフォルトです。追加モードでは、Extract は異常終了時、証跡にリカバリ・マーカーを書き込みます。Extract が再起動され、リカバリ・マーカーを検出すると、ローカル・トランザクションが有効な場合、不完全なトランザクションのロールバックをリクエストします。ローカル・トランザクションが無効な場合、警告メッセージが発行されます。プロパティ gg.handler.{name}.localTX が明示的に false に設定されていないかぎり、ローカル・トランザクションは有効です。

レポートの問題

Oracle GoldenGate のサポート・アカウントがある場合、サポート・チケットを発行してください。次のものを含めてください。

- オペレーティング・システムと Java のバージョン

  Java Runtime Environment のバージョンは、次のようにすると表示できます。

  ```
  $ java -version
  ```

- 構成ファイル
  - ユーザー・イグジットを実行する Extract のパラメータ・ファイル
  - JMS または JNDI プロパティ・ファイルを含む、使用されるすべてのプロパティ・ファイル
  - ユーザー・イグジットの Velocity テンプレート

- ログ・ファイル

  Oracle GoldenGate のインストール・ディレクトリ内のすべての .log ファイル (Java log4j ログ・ファイルおよびユーザー・イグジットまたは VAM のログ・ファイル)。

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

L

M

O

P

S

T

U

V

W

X

ア

イ

エ

カ

キ

ク

コ

サ

シ

セ

ソ

タ

チ

テ

ト

ハ

ヒ

フ

へ

メ

モ

ユ

レ

ロ