BEA WebLogic Server™

インターナショナライゼーション ガイド

BEA WebLogic Server バージョン 7.0
マニュアルの改訂 : 2002 年 8 月 20 日

著作権

Copyright © 2002, BEA Systems, Inc. All Rights Reserved.

限定的権利条項

本ソフトウェアおよびマニュアルは、BEA Systems, Inc. 又は日本ビー・イー・エー・システムズ株式会社 (以下、「BEA」といいます) の使用許諾契約に基づいて提供され、その内容に同意する場合にのみ使用することができ、同契約の条項通りにのみ使用またはコピーすることができます。同契約で明示的に許可されている以外の方法で同ソフトウェアをコピーすることは法律に違反します。このマニュアルの一部または全部を、BEA からの書面による事前の同意なしに、複写、複製、翻訳、あるいはいかなる電子媒体または機械可読形式への変換も行うことはできません。

米国政府による使用、複製もしくは開示は、BEA の使用許諾契約、および FAR 52.227-19 の「Commercial Computer Software-Restricted Rights」条項のサブパラグラフ (c)(1)、DFARS 252.227-7013 の「Rights in Technical Data and Computer Software」条項のサブパラグラフ (c)(1)(ii) 、NASA FAR 補遺 16-52.227-86 の「Commercial Computer Software--Licensing」条項のサブパラグラフ (d)、もしくはそれらと同等の条項で定める制限の対象となります。

このマニュアルに記載されている内容は予告なく変更されることがあり、また BEA による責務を意味するものではありません。本ソフトウェアおよびマニュアルは「現状のまま」提供され、商品性や特定用途への適合性を始めとする (ただし、これらには限定されない) いかなる種類の保証も与えません。さらに、BEA は、正当性、正確さ、信頼性などについて、本ソフトウェアまたはマニュアルの使用もしくは使用結果に関していかなる確約、保証、あるいは表明も行いません。

商標または登録商標

BEA, Jolt, Tuxedo, および WebLogic は BEA Systems, Inc. の登録商標です。BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop および How Business Becomes E-Business は、BEA Systems, Inc の商標です。

その他の商標はすべて、関係各社がその権利を有します。

インターナショナライゼーション ガイド

パート番号	マニュアルの改訂	ソフトウェアのバージョン
なし	2002 年 8 月 20 日	BEA WebLogic Server バージョン 7.0

目次

3. BEA WebLogic Server メッセージ エディタの使い方

4. BEA WebLogic Server インターナショナライゼーション ユーティリティの使い方

A. BEA WebLogic Server 用の Localizer クラスのリファレンス

B. BEA WebLogic Server 用の Logger クラスのリファレンス

C. BEA WebLogic Server 用の Loggable オブジェクトのリファレンス

D. BEA WebLogic Server 用の TextFormatter クラスのリファレンス

このマニュアルの内容

このマニュアルでは、インターナショナライゼーションとローカライゼーションについて定義し、WebLogic Server で提供されるテンプレートおよびツールを使用して、ロケール固有のメッセージ カタログを作成または編集する方法について説明します。

このマニュアルの構成は次のとおりです。

- 第 1 章「WebLogic Server のインターナショナライゼーションの概要」では、インターナショナライゼーションとローカライゼーションに必要なプロセスの概要について説明します。
- 第 2 章「BEA WebLogic Server でのメッセージ カタログの使い方」では、メッセージ カタログの種類と、メッセージ定義、要素、および引数について説明します。
- 第 3 章「BEA WebLogic Server メッセージ エディタの使い方」では、WebLogic Server に付属の Message Editor の使用方法について説明します。
- 第 4 章「BEA WebLogic Server インターナショナライゼーション ユーティリティの使い方」では、WebLogic Server に付属のインターナショナライゼーション ユーティリティの使用方法について説明します。
- 付録 A「BEA WebLogic Server 用の Localizer クラスのリファレンス」では、`Localizer` クラス、`Localizer` メソッド、`Localizer` のキー値、および `Localizer` のルックアップ プロパティについて説明します。
- 付録 B「BEA WebLogic Server 用の Logger クラスのリファレンス」では、`Logger` クラスについて説明し、メッセージ カタログおよび対応する `Logger` クラスのサンプルを示します。
- 付録 C「BEA WebLogic Server 用の Loggable オブジェクトのリファレンス」では、`loggable objects` および使用方法について説明します。
- 付録 D「BEA WebLogic Server 用の TextFormatter クラスのリファレンス」では、`TextFormatter` クラスを使用するアプリケーションのサンプルを示します。

対象読者

このマニュアルは、ロケール固有の管理のために、WLS 配布キットに含まれるメッセージ カタログをインターナショナライズまたはローカライズする必要があるアプリケーション開発者を主な対象としています。WebLogic Server プラットフォームを読者がよく理解し、Web 技術、オブジェクト指向プログラミング技術、および Java プログラミング言語にも精通していることを前提として書かれています。

e-docs Web サイト

BEA 製品のドキュメントは、BEA の Web サイトで入手できます。BEA のホーム ページで [製品のドキュメント] をクリックします。

このマニュアルの印刷方法

Web ブラウザの [ファイル | 印刷] オプションを使用すると、Web ブラウザからこのマニュアルを一度に 1 章ずつ印刷できます。

このマニュアルの PDF 版は、WebLogic Server の Web サイトで入手できます。PDF を Adobe Acrobat Reader で開くと、マニュアルの全体 (または一部分) を書籍の形式で印刷できます。PDF を表示するには、WebLogic Server ドキュメントのホーム ページを開き、[ドキュメントのダウンロード] をクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader は Adobe の Web サイト (http://www.adobe.co.jp) で無料で入手できます。

関連情報

インターナショナリゼーションおよびローカライゼーションの一般情報については、以下のソースを参照してください。

- `java.sun.com` の Java Developer ConnectionTM
- `http://www.w3.org` の World Wide Web Consortium (W3C) Web Site にある Internationalization セクション

サポート情報

BEA のドキュメントに関するユーザからのフィードバックは弊社にとって非常に重要です。質問や意見などがあれば、電子メールで docsupport-jp@beasys.com までお送りください。寄せられた意見については、ドキュメントを作成および改訂する BEA の専門の担当者が直に目を通します。

電子メールのメッセージには、ご使用のソフトウェアの名前とバージョン、およびドキュメントのタイトルと日付をお書き添えください。本バージョンの BEA WebLogic Server について不明な点がある場合、または BEA WebLogic Server のインストールおよび動作に問題がある場合は、BEA WebSupport (www.bea.com) を通じて BEA カスタマ サポートまでお問い合わせください。カスタマ サポートへの連絡方法については、製品パッケージに同梱されているカスタマ サポート カードにも記載されています。

カスタマ サポートでは以下の情報をお尋ねしますので、お問い合わせの際はあらかじめご用意ください。

- お名前、電子メール アドレス、電話番号、ファクス番号
- 会社の名前と住所
- お使いの機種とコード番号
- 製品の名前とバージョン
- 問題の状況と表示されるエラー メッセージの内容

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
〔Ctrl〕+ 〔Tab〕	複数のキーを同時に押すことを示す。
斜体	強調または書籍のタイトルを示す。

等幅テキスト	コード サンプル、コマンドとそのオプション、Java クラス、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。
	例:
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

斜体の等幅テキスト	コード内の変数を示す。 例: `String *CustomerName*;`
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 例: LPT1 BEA_HOME OR
{ }	構文の中で複数の選択肢を示す。

表記法	適用
[ ]	構文の中で任意指定の項目を示す。 例： `java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]`
|	構文の中で相互に排他的な選択肢を区切る。 例： `java weblogic.deploy [list|deploy|undeploy|update] password {application} {source}`
...	コマンドラインで以下のいずれかを示す。 ■ 引数を複数回繰り返すことができる ■ 任意指定の引数が省略されている ■ パラメータや値などの情報を追加入力できる。

1 WebLogic Server のインターナショナライゼーションの概要

以下の節では、ローカライゼーションおよびインターナショナライゼーションの概要について説明します。

- インターナショナライゼーションおよびローカライゼーションの標準規格
- WebLogic Server のローカライゼーション
- メッセージ カタログ
- インターナショナライゼーション用 Java インタフェース
- インターナショナライズされたメッセージを作成する主要な手順

インターナショナライゼーションおよびローカライゼーションの標準規格

BEA では、W3C (World Wide Web Consortium: WWW コンソーシアム) 推奨による、世界中のすべての言語および記述システムで使用できる標準フォーマットおよびプロトコルを採用しています。これらの規格は、WebLogic Server で使用されている Java インターナショナライゼーション API (Application Program Interfaces: アプリケーション プログラム インタフェース) に組み込まれています。インターナショナライゼーション (I18N) とは、さまざまな地域で適切に動作するようにソフトウェアを準備することです。ローカライゼーション (L10N) とは、実行時にロケール固有の言語および構成を使用することです。

WebLogic Server 内のテキスト データのインターナショナライゼーションは、メッセージ カタログによって提供されます。WebLogic Server では、シンプル テキストだけでなくログ メッセージのメッセージ カタログもサポートしています。ログ メッセージには、ログ ファイルに書き込まれるデータが含まれています。この極めて動的なデータには、アプリケーションやシステムの現在の状態に特定

の情報が含まれています。ローカライズされたログ メッセージ カタログのテキストと結合すれば、このデータはユーザの言語でエラー状態を説明する、適切にフォーマットされたローカライズ済みのメッセージになります。コンソールに送信される出力は、シンプル テキストです。ログ メッセージと同様、シンプル テキストも動的データと結合して使用できます。

このガイドでは、テキスト データのインターナショナライゼーションに関連する情報だけを提供します。

WebLogic Server のローカライゼーション

ローカライゼーションでは、言語だけでなく、照合、日付と時刻の書式、通貨の書式、および文字コードも対象となります。WebLogic Server のエラー ログに記録されるメッセージは、ユーザ個々の要件に合わせてローカライズできます。

WebLogic Server のインターナショナライゼーションでは、以下の 2 種類のデータのローカライゼーションがサポートされます。

- **ログ メッセージ** — ログ メッセージはサーバ ログに書き込まれる通知メッセージであり、適切なメッセージ引数がメッセージの定義で指定されている場合はエラー メッセージも含まれます。
- **シンプル テキスト** — サーバで表示しなければならないログ メッセージや例外以外のテキスト (ユーティリティからの出力など)。シンプル テキストの例としてはヘルプ メッセージ、GUI (Graphical User Interface: グラフィカル ユーザ インタフェース) ラベル、エラー メッセージなどがあります。

メッセージ カタログ

インターナショナライズするテキストはすべてメッセージ カタログで定義します。各メッセージ カタログでは、ログ メッセージまたはシンプル テキストのコレクションが定義されます。インターナショナライズされたメッセージを作成するには、コードの変更や再コンパイルをせずに、さまざまなロケールに文字列を簡単に変換できるように、すべてのメッセージ文字列をメッセージ カタログに外部化する必要があります。アプリケーション コードは、ロギング メソッドの

実行時値を提供します。ロギング メソッドは、現在のロケールに従ってコードとカタログ内のメッセージ文字列を結合します。次に、ローカライズされたメッセージをアプリケーション コードがログ ファイルに出力します。

メッセージ カタログには、次の 3 つのタイプがあります。

- **ログ メッセージ カタログ—**ログ メッセージのコレクション。
- **シンプル テキスト カタログ—**シンプル テキスト メッセージのコレクション。
- **ロケール メッセージ カタログ—**最上位のログ メッセージまたはシンプル テキスト メッセージに対応するロケール固有のメッセージのコレクション。

ログ メッセージまたはロケール メッセージ カタログのメッセージ ID は、すべてのログ メッセージないしロケール メッセージ カタログでユニークです。メッセージ カタログ ファイル内で、各メッセージのローカライズ版には、エラーに固有のユニークなメッセージ ID とメッセージ テキストが割り当てられます。原則的に、メッセージは、サポート担当者が簡単に見つけられるように、システム内の 1 つの場所からログに記録します。シンプル テキスト カタログのメッセージ ID は、個別のシンプル テキスト カタログ内でユニークになっています。

メッセージ カタログの詳細については、2-1 ページの「BEA WebLogic Server でのメッセージ カタログの使い方」を参照してください。

インターナショナライゼーション用 Java インタフェース

WebLogic Server では、インターナショナライゼーションおよびローカライゼーションに Java インターナショナライゼーション インタフェースを使用しています。ユーザは、WebLogic Server でインターナショナライゼーションをどのように処理するかを理解するだけでなく、この Java インターナショナライゼーション インタフェースおよび JDK (Java Development Kit: Java 開発キット) に組み込まれている以下のクラスに関する知識も必要です。

クラス	説明
java.util.Locale	地理的、政治的、または文化的に区別される特定の地域を表す。
java.util.ResourceBundle	ロケール固有のオブジェクトが格納されるコンテナを提供する。
java.text.MessageFormat	言語に依存しない方法で連結されたメッセージを生成する。

インターナショナライズされたメッセージを作成する主要な手順

以下の手順は、WebLogic Server で使用する、インターナショナライズされたメッセージを作成する方法を示しています。

1. カタログ内のメッセージを定義して、最上位のログ カタログまたはシンプル テキスト カタログを作成、あるいは編集します。詳細については、3-1 ページの「BEA WebLogic Server メッセージ エディタの使い方」を参照してください。

   メッセージ テキストだけでなく、メッセージが格納する実行時値の型と配置に関する情報も含みます。

2. `i18ngen` を実行して、手順 1 で作成または編集したカタログを検証し、実行時クラスを生成します。

   生成されたクラスには、各メッセージで使用するメソッドが組み込まれています。このメソッドは、メッセージ カタログ エントリで指定されたデータに従って定義されます。メソッドには、カタログのタイプによって、`Logger` または `TextFormatter` メソッドが組み込まれています。詳細については、4-4 ページの「18ngen ユーティリティ」を参照してください。

3. 手順 1 で作成したメッセージ カタログの必要に応じてロケール固有カタログを作成します。

4. `l10ngen` を実行して、ロケール固有のカタログを処理します。

5. 手順 2 で生成した `Logger` または `TextFormatter` メソッドを使用できるようにアプリケーションを設定します。アプリケーションがメッセージをログに書き込む、または返す場合、メッセージは使用された `Logger` または `TextFormatter` メソッドによりローカライズされたバージョンのテキストを用いて書き込まれます。

これらの手順は、以下のトピックで詳細に説明されています。ロギング サブシステムの概要やログ メッセージの要素の説明を含めた詳細については、『管理者ガイド』の「ログ メッセージを使用した WebLogic Server の管理」を参照してください。

2 BEA WebLogic Server でのメッセージ カタログの使い方

以下の節では、メッセージ カタログとその使い方について説明します。

- メッセージ カタログの概要
- メッセージ カタログの階層
- メッセージ カタログの名前の選択
- メッセージ引数の使い方
- メッセージ カタログのフォーマット

メッセージ カタログの概要

メッセージ カタログは、テキスト メッセージのコレクションの説明が格納される XML ファイルです。各メッセージは、ユニークな識別子でインデックス付けされます。これらの XML ファイルを、`i18ngen` ユーティリティのビルド プロセスでクラスにコンパイルします (詳細については、4-4 ページの「18ngen ユーティリティ」を参照してください)。生成されたクラスのメソッドは、実行時にメッセージをログに記録するために使用されるオブジェクトとなります。

メッセージ カタログでは、複数のロケールまたは言語がサポートされています。個々のメッセージ カタログには、最上位カタログというデフォルト バージョンが必ず 1 つあります。さらに、追加してサポートされるロケールごとに、対応するロケール固有のカタログが付加されます。最上位カタログには、メッセージの定義に必要なすべての情報が含まれます。ロケール固有のカタログには、メッセージ ID、変更された日付、および特定ロケールに対応するメッセージの変換だけが含まれます。

メッセージ カタログ ファイルは、XML DTD (Document Type Definition: 文書型定義) によって定義されます。この DTD は msgcat ディレクトリに格納されます。msgcat ディレクトリの場所は、WebLogic Server のインストール先によって異なる場合があります。デフォルト インストール パスを使用した場合、msgcat ディレクトリは *WL_HOME*\samples\server\src\examples\i18n\msgcat の下の BEA_HOME ディレクトリにあります。

WebLogic Server 配布キットには、以下の 2 つの DTD が付属しています。

- msgcat.dtd ―最上位のデフォルト カタログの構文を記述する。
- l10n_msgcat.dtd ―ロケール固有のカタログの構文を記述する。

この msgcat ディレクトリには、最上位のメッセージ カタログおよびロケール固有のメッセージ カタログ作成に使用できるテンプレートも入っています。

メッセージ カタログの作成では、ロギングのすべての必要条件に対応するログ メッセージ カタログを 1 つ作成するか、サブシステムまたは Java パッケージごとに細かく分けてカタログを作成するか、いずれかを選択できます。複数のサブシステムを使用すると、表示するときにそのログの特定の部分に焦点を当てることができるので、この方法をお勧めします。

シンプル テキストのカタログの場合は、インターナショナライズされるユーティリティごとに 1 つのカタログを作成することをお勧めします。3-1 ページの「BEA WebLogic Server メッセージ エディタの使い方」で説明されているメッセージ エディタを使用すれば、特定サイト用のメッセージ カタログを作成することができます。

メッセージ カタログの階層

すべてのメッセージおよび例外は、デフォルトの最上位カタログで定義する必要があります。WebLogic Server 配布キットには、*WL_HOME*\samples\server\src\examples\i18n\msgcat ディレクトリにサンプル カタログ集が付属しています。

注意 : このディレクトリ パスは、WebLogic Server のインストール先によっては異なる場合があります。

基本カタログのさまざまなローカライゼーションを提供するカタログは、ロケールに基づいた名前がついた msgcat のサブディレクトリ (例 : ドイツの場合 msgcat\de) に定義されています。たとえば、mycat.xml という最上位カタログと ..\de\mycat.xml というそのドイツ語版カタログがある、といった要領です。通常、最上位カタログは英語ですが、インストールされている WebLogic Server のカタログ以外のカタログで英語は必要ではありません。

java.util.Locale のドキュメントで定義されているように、ロケールの指定 (de など) にも階層があります。ロケールでは言語、国、およびバリアントを指定できます。言語は、ロケール指定でもっとも一般的なものです。言語は国コードによって拡張できます。たとえば、en\US はアメリカ英語を示します。これに関連付けられたカタログの名前は、..\en\US\mycat.xml となります。バリアントはベンダまたはブラウザに固有であり、言語または国で定義された複数のロケールの間で細かな違い (照合順序など) を持たせるために使用します。

メッセージ カタログの名前の選択

メッセージ カタログ ファイルの名前 (.xml 拡張子なし) は、実行時クラスおよびプロパティ名の生成に使用されるため、命名は慎重に行なってください。

メッセージ カタログに名前をつける場合、以下のガイドラインに従ってください。

- 対象パッケージ内の既存クラスの名前と衝突するようなメッセージ カタログ名は避ける。
- メッセージ カタログ名に含まれる文字は、クラス名に使用できる文字だけを使用する。
- クラス名の命名規約に従う。

たとえば、カタログの名前が Xyz.xml の場合、生成されるクラスの名前は XyzLogLocalizer および XyzLogger になります。

以下に挙げる考慮事項も、メッセージ カタログ ファイルに適用されます。

- メッセージ ID は、通常は先頭に 0 が付く 6 文字の文字列です。一部のインタフェースでは整数表現もサポートされています。

- Java では、パッケージと呼ばれるコレクションにクラスをまとめることができます。パッケージ名は、特定のカタログが存在するサブシステムの名前と一致している必要があります。
- ログの Localizer「クラス」は、実際には ResourceBundle プロパティ ファイルです。

メッセージ引数の使い方

ログ メッセージのメッセージ本文、メッセージの詳細、原因、およびアクションといったセクションには、java.text.MessageFormat に記載されているようなメッセージ引数を組み込むことができます。シンプル メッセージでは、引数を入れられるのは、メッセージ本文のセクションに限られます。引数は、実行時に動的に設定される値を表します。これらの値は、メッセージを出力するなど、ルーチンに渡されます。1 つのメッセージには 0 ～ 9 で番号付けされた 10 個までの引数をサポートできます。メッセージ本文にはすべての引数を入れるようにしますが、メッセージ定義の任意のテキスト セクションにもこれらの引数の任意のサブセットを入れることができます。メッセージ引数は開発時にメッセージの定義に挿入され、実行時においてメッセージがログに記録されるときに適切なメッセージ コンテンツに置き換えられます。

以下は、XML ログ メッセージ定義の抜粋で、メッセージ引数の使い方を示しています。引数の番号は、*method* 属性で指定された引数の 1 つと対応していなければなりません。具体的には、{0} は最初の引数、{1} は 2 番目の引数、という要領で対応している必要があります。

コード リスト 2-1 メッセージ引数の例

```
<messagebody>Unable to open file, {0}.</messagebody>
     <messagedetail>
     File, {0} does not exist. The server will restore the file
     contents from {1}, resulting in the use of default values
     for all future requests.
     </messagedetail>
     <cause>The file was deleted</cause>
     <action>
     If this error repeats then investigate unauthorized access to
     the file system.
     </action>
```

次に、上のメッセージの method 属性の例を示します。

```
-method=゛logNoFile(String name, String path)"
```

このメッセージでは、2 つの引数 {0} と {1} が使用されます。

- {0} がメッセージ本文で使用されています。
- メッセージの詳細では両方の引数が使用されています。
- <cause> セクションまたは <action> セクションではどちらも使用されていません。

また、引数は文字列であるか、文字列として表現可能である必要があります。数値データは {n,number} として表されます。日付は {n,date} の形式がサポートされます。また、ログ メッセージでは重要度レベルを割り当てる必要があります。ログ メッセージは、Logger メソッドによって method 属性で定義されているとおりに生成されます。

メッセージ カタログのフォーマット

最上位のカタログ ファイルとロケール固有のカタログ ファイルでは、カタログのフォーマットが若干異なります。最上位のカタログでは、基本ロケールのテキスト メッセージが定義されます。ロケール固有のカタログは、最上位バージョンで定義されているテキストの翻訳だけを備えています。さらに、ログ メッセージ カタログではシンプル テキスト カタログと定義の方法が異なります。

これら各タイプのカタログの要素は、以下の節で説明されています。

- ログ メッセージ カタログの要素
- シンプル テキスト メッセージ カタログの要素
- ロケール固有のカタログの要素

ログ メッセージ カタログの要素

この節では、ログ メッセージ カタログの以下の要素に関するリファレンス情報を提供します。

- message_catalog
- log_message
- その他の log_message カタログ要素

message_catalog

次の表では、`message_catalog` 要素で定義できる各属性について説明します。

属性	デフォルト値	必須 / 省略可能	説明
`i18n_package`	`weblogic.i18n`	省略可能	このカタログの Logger クラスを含む Java パッケージ。クラスの名前はカタログ ファイルの名前に基づいて付けられる。たとえばカタログの名前が `mycat.xml` の場合、生成される Logger クラスの名前は *`i18n_package`*`.mycatLogger.class` になる。
`l10n_package`	`weblogic.i18n`	省略可能	このカタログの `LogLocalizer` プロパティを含む Java パッケージ。クラスの名前はカタログ ファイルの名前に基づいて付けられる。たとえばカタログの名前が `mycat.xml` の場合は、*`l10n_package`*`.mycatLogLocalizer.properties` という名前のプロパティ ファイルが生成される。
`subsystem`	なし	必須	このカタログに関連付けられているサブシステムを識別する頭字語。サブシステムの名前はエラー ログに含まれ、メッセージの分離を目的として使用される。

属性	デフォルト値	必須 / 省略可能	説明
version	なし	必須	使用される msgcat.dtd のバージョンを指定する。書式は *n.n* で、たとえば version="1.0" のように定義する。数値は「1.0」以上で指定する。
baseid	000000: WebLogic Server カタログ 500000 ユーザ定義のカタログ	省略可能	このカタログで使用される最小のメッセージ ID を指定する。構文は 0 ～ 9 を 6 つ並べたもの。
endid	499999: WebLogic Server カタログ 999999: ユーザ定義のカタログ	省略可能	このカタログで使用される最大のメッセージ ID を指定する。構文は 0 ～ 9 を 6 つ並べたもの。

log_message

次の表では、log_message 要素で定義できる各属性について説明します。

属性	デフォルト値	必須 / 省略可能	説明
messageid	なし	必須	このログ メッセージのユニークな識別子。識別子は、すべてのカタログにわたってユニークでなければならない。値は、baseid 属性と endid 属性で定義された範囲で指定する必要がある。これは message_catalog の子要素となっている。

属性	デフォルト値	必須 / 省略可能	説明
datelastchanged	なし	省略可能	このメッセージの修正管理に使用される日付と時刻のスタンプ。日付は、カタログで動作するユーティリティによって提供される。構文は次のとおり。 `Long.toString(new Date().getTime());`
severity	なし	必須	ログ メッセージの重要度を示す。値は、debug、info、warning、notice、error、critical、alert、または emergency のいずれか。ユーザ カタログで使用できるのは、debug、info、warning および error に限られる。
stacktrace	true	省略可能	Throwable の引数に対するスタック トレースを生成するかどうかを示す。指定できる値は true または false。値が true の場合はトレースが生成される。構文は次のとおり。 `stacktrace="true"`

属性	デフォルト値	必須 / 省略可能	説明
`method`	なし	必須	このメッセージをログに記録するためのメソッド シグネチャ。実際には、ここで指定されるメソッドと Throwable 引数が追加された同様のメソッドの 2 つが提供される。 構文は標準の Java メソッド シグネチャから修飾子、セミコロン、および拡張子を除いたもの。引数の型には Java プリミティブまたは Java クラスを使用できる。クラスは `java.lang` にない場合は完全修飾でなければならない。また、クラスは `java.text.MessageFormat` の規約に準拠していなければならない。たいていの場合、クラス引数は便利な `toString()` メソッドを備えている。 引数には有効であればどのような名前でも指定できるが、`argn` (n は 0 ～ 9) の規約に従っていなければならない。指定できる引数は 10 個まで。各 `argn` について、2-9 ページの「その他の log_message カタログ要素」で説明されているテキスト要素に少なくとも 1 つの対応するプレースホルダが存在しなければならない。プレースホルダの形式は `{n}`、`{n,number}`、または `{n,date}`。
`loggables`	`False`	省略可能	各メッセージに対して、Loggable オブジェクトを返すメソッドを生成するかどうかを指定する。構文は次のとおり。 `loggables="true"` C-1 ページの「BEA WebLogic Server 用の Loggable オブジェクトのリファレンス」を参照

その他の log_message カタログ要素

次の表では、log_message 要素の子要素を説明します。

要素	親要素	必須 / 省略可能	説明
messagebody	log_message	必須	このメッセージの短い説明を示す文字列。この要素では 0 個以上のプレースホルダ {n} を使用できる。プレースホルダは、ログ メッセージがローカライズされるときに適切な引数で置き換えられる。
messagedetail	log_message	必須	イベントの詳しい説明を示す文字列。この要素では 0 個以上のプレースホルダ {n} を使用できる。プレースホルダは、ログ メッセージがローカライズされるときに適切な引数で置き換えられる。
cause	log_message	必須	問題の根本的な原因を説明する文字列。この要素では 0 個以上のプレースホルダ {n} を使用できる。プレースホルダは、ログ メッセージがローカライズされるときに適切な引数で置き換えられる。
action	log_message	必須	適切な解決策を説明する文字列。この要素では 0 個以上のプレースホルダ {n} を使用できる。プレースホルダは、ログ メッセージがローカライズされるときに適切な引数で置き換えられる。

ログ メッセージ カタログの例

次の例は、ログ メッセージを 1 つ持つログ メッセージ カタログ MyUtilLog.xml を示しています。

コード リスト 2-2 ログ メッセージ カタログの例

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls700/dtd/msgcat.dtd">
<message_catalog
  l10n_package="programs.utils"
  i18n_package="programs.utils"
```

```
  subsystem="MYUTIL"
  version="1.0"
  baseid="600000"
  endid="600100"
  <log_message
    messageid="600001"
    severity="warning"
    method="logNoAuthorization(String arg0, java.util.Date arg1,
       int arg2)"
    <messagebody>
     Could not open file, {0} on {1,date} after {2,number} attempts.
    </messagebody>
    <messagedetail>
      The configuration for this application will be defaulted to
      factory settings. Custom configuration information resides
      in file, {0}, created on {1,date}, but is not readable.
    </messagedetail>
    <cause>
     The user is not authorized to use custom configurations. Custom
     configuration information resides in file, {0}, created on
     {1,date}, but is not readable.The attempt has been logged to
     the security log.
    </cause>
    <action>
      The user needs to gain approriate authorization or learn to
      live with the default settings.
    </action>
  </log_message>
</message_catalog>
```

シンプル テキスト メッセージ カタログの要素

この節では、シンプル テキスト メッセージ カタログの以下の要素に関するリファレンス情報を提供します。

- message_catalog
- message
- messagebody

message_catalog

次の表では、message_catalog 要素で定義できる各属性について説明します。

属性	デフォルト値	必須 / 省略可能	説明
l10n_package	weblogic.i18n	省略可能	このカタログに生成された TextFormatter クラスおよび TextLocalizer プロパティを含む Java パッケージ。クラスの名前はカタログ ファイルの名前に基づいて付けられる。 たとえばカタログの名前が mycat.xml の場合は、l10n_package.mycatTextLocalizer.properties という名前のプロパティ ファイルが生成される。
subsystem	なし	必須	このカタログと関連付けられているサブシステムを識別する頭字語。サブシステムの名前はエラー ログに含まれ、メッセージの分離を目的として使用される。
version	なし	必須	使用される msgcat.dtd のバージョンを指定する。書式は *n.n* で、たとえば version="1.0" のように定義する。数値は「1.0」以上でなければならない。

message

次の表では、message 要素で定義できる各属性について説明します。

属性	デフォルト値	必須 / 省略可能	説明
messageid	なし	必須	このログ メッセージのユニークな識別子 (英数字の文字列)。識別子はこのカタログのコンテキストの中でユニークであれば良い。message は message_catalog の子要素である。

属性	デフォルト値	必須 / 省略可能	説明
datelastchanged	なし	省略可能	このメッセージの修正の管理に便利な日付と時刻のスタンプ。

method	なし	省略可能	このメッセージをフォーマットするためのメソッド シグネチャ。
			構文は標準の Java メソッド シグネチャから戻り値の型、修飾子、セミコロン、および拡張子を除いたもの。戻り値の型は常に String。引数の型には Javaプリミティブまたは Java クラスを使用できる。クラスは java.lang にない場合は完全修飾でなければならない。また、クラスは java.text.MessageFormat の規約に準拠していなければならない。たいていのクラス引数は便利な toString() メソッドを備えており、対応する MessageFormat プレースホルダは文字列でなければならない (つまり {n} の形式)。引数には有効であればどのような名前でも指定できる。指定できる引数は 10 個まで。
			各引数について、次に説明する messagebody 要素に少なくとも 1 つの対応するプレースホルダが存在しなくてはならない。プレースホルダの形式は {n}、{n,number}、または {n,date}。
			例 :
			`method="getNoAuthorization (String filename, java.util.Date creDate)"`
			この例は、次のように TextFormatter クラスのメソッドになる。
			`public String getNoAuthorization (String filename, java.util.Date creDate)`

messagebody

次の表では、message 要素の子要素を説明します。

要素	親要素	必須 / 省略可能	説明
messagebody	メッセージ	必須	メッセージと関連付けられたテキスト。この要素では、0 個以上のプレースホルダ {n} を使用できます。プレースホルダは、ログ メッセージがローカライズされるときに適切な引数で置き換えられます。

シンプル テキスト カタログの例

次の例は、テキスト定義が 1 つあるシンプル テキスト カタログ MyUtilLabels.xml を示しています。

コード リスト 2-3 シンプル テキスト カタログの例

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
    "http://www.bea.com/servers/wls700/dtd/msgcat.dtd">
<message_catalog>
 l10n_package="programs.utils"
 i18n_package="programs.utils"
 subsystem="MYUTIL"
 version="1.0"
 <message>
  messageid="FileMenuTitle"
  <messagebody>
    File
  </messagebody>
 </message>
</message_catalog>
```

ロケール固有のカタログの要素

ロケール固有のカタログは、最上位カタログのサブセットです。ロケール固有のカタログは、それらが表すロケールに基づいた名前のサブディレクトリに格納されます。以下の節で説明する要素および属性は、ロケール固有のカタログに有効なものです。

locale_message_catalog

次の表では、`locale_message_catalog` 要素で定義できる各属性について説明します。

属性	デフォルト値	必須 / 省略可能	説明
`l10n_package`	`weblogic.i18n`	省略可能	このカタログに対して生成された `LogLocalizer` または `TextLocalizer` プロパティを含む Java パッケージ。プロパティの名前はカタログ ファイル名に基づいて設定される。 たとえば、`mycat.xml` という名前のフランス語版のログ メッセージ カタログでは、プロパティ ファイルは `l10n_package.mycatLogLocalizer_fr_FR.properties` となる。
`version`	なし	必須	使用される `msgcat.dtd` のバージョンを指定する。書式は *`n.n`* で、たとえば `version="1.0"` のように定義する。数値は「1.0」以上でなければならない。

log_message

ロケール固有のカタログでは、ログ メッセージ カタログの `log_message` 要素で定義された属性が使用されますので、この要素は改めて定義する必要はありません。

その他の locale_message_catalog 要素

ロケール固有のカタログでは、ログ メッセージ カタログで定義された messagebody、messagedetail、cause、および action の各カタログ要素を使用しますので、これらの要素を改めて定義する必要はありません。

ロケール メッセージ カタログの構文

次の例は、...\msgcat\fr\MyUtilLabels.xml にあるメッセージのフランス語版です。

翻訳されたメッセージをコード リスト 2-4 に示します。

コード リスト 2-4 フランス語に翻訳されたメッセージの例

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC
   "weblogic-locale-message-catalog-dtd"
   "http://www.bea.com/servers/wls700/dtd/l10n_msgcat.dtd">
<locale_message_catalog
  l10n_package="programs.utils"
  subsystem="MYUTIL"
  version="1.0">
  <message>
    <messageid="FileMenuTitle">
    <messagebody> Fichier </messagebody>
  </message>
</locale_message_catalog>
```

messagebody、messagedetail、cause、および action の各要素でテキストを入力するときには、有効な UTF-8 (Unicode Transformation Format-8) 文字を生成するツールを使用するとともに、適切なキーボードのマッピングがインストールされている必要があります。UTF-8 は、ASCII 文字のエンコーディングを最適化する、効率的な Unicode 文字列エンコーディング方式です。メッセージ カタログでは常に UTF-8 エンコーディング方式が使用されます。WebLogic Server と共にダウンロードされる MessageLocalizer ユーティリティは、有効な UTF-8 文字の生成に使用できるツールです。

3 BEA WebLogic Server メッセージ エディタの使い方

以下の節では、メッセージ エディタの使用方法について説明します。

- メッセージ エディタの概要
- メッセージ エディタの起動
- カタログに関する作業
- カタログへのメッセージの追加
- メッセージの検索
- Message Viewer の使い方
- 既存のメッセージの編集

メッセージ エディタの概要

メッセージ エディタは、XML メッセージ カタログの作成、読み込み、および書き込みに使用するグラフィカル インタフェース ツールです。メッセージ エディタは、WebLogic Server のインストール時にインストールされます。必要に応じて、XML カタログはテキスト エディタで直接に編集することもできます。

注意： メッセージ エディタでは、ローカライズされたカタログの編集はできません。

メッセージ エディタで可能な作業は以下のとおりです。

- XML メッセージ カタログを作成する
- メッセージを作成および編集する

- 1 つのカタログのすべてのメッセージを表示する
- 複数のカタログのすべてのメッセージを同時に表示する
- メッセージを検索する
- カタログ エントリの XML の有効性を検証する

メッセージ エディタで作成または使用されている最中のカタログは、コンテキスト カタログと言います。

カタログ パーサ (`i18ngen` または `l10ngen`) は、特殊文字に変換する文字コード参照を認識し、表 3-1 に示すように対象文字を目的の特殊文字に変換します。メッセージ エディタは、これらの特殊文字を認識し、文字コード参照を使用してまた書き込みます。パーサでは、他の文字参照は認識されません。

表 3-1 特殊文字の参照

文字コード	特殊文字
`&`	@
`<`	<
`>`	>
`&apos;`	‘
`&quot;`	“

メッセージ エディタの起動

メッセージ エディタを起動する前に、WebLogic Server システムをインストール、コンフィグレーションし、環境変数 (`setExamplesEnv.cmd`) の設定を完了している必要があります。また、クラスパスの設定が正しいことを確認してください。

サンプルのメッセージ カタログ ファイルは、`BEA_HOME\samples\wlserver7.0\samples\examples\i18n\msgcat` ディレクトリにあります。

注意： このディレクトリ パスは、WebLogic Server のインストール先によっては異なる場合があります。

メッセージ エディタを起動するには、次のように入力します。

```
java weblogic.MsgEditor
```

または

```
java weblogic.i18ntools.gui.MessageEditor
```

ログ メッセージ用の WebLogic メッセージ エディタのメイン ウィンドウは図 3-1 のようになります。

図 3-1 ログ メッセージ用 WebLogic メッセージ エディタ

カタログに関する作業

以下の節では、メッセージ エディタをカタログ管理に使用する方法について説明します。

- 既存のカタログの参照
- 新規カタログの作成

既存のカタログの参照

WebLogic メッセージ エディタのメイン ウィンドウから、既存のカタログを検索するには、[**Message Catalog**] フィールドに絶対パス名を入力するか、[**Browse**] をクリックして [Open] ダイアログから既存のカタログに移動します。

図 3-2 [Open] ダイアログ

WebLogic Server のインストールに付属しているサンプル カタログは、`BEA_HOME\samples\wlserver7.0\samples\examples\i18n\msgcat` ディレクトリにあります。

注意 : このディレクトリ パスは、WebLogic Server のインストール先によっては異なる場合があります。

ユーザ定義のカタログは、任意のディレクトリを指定して配置できます。

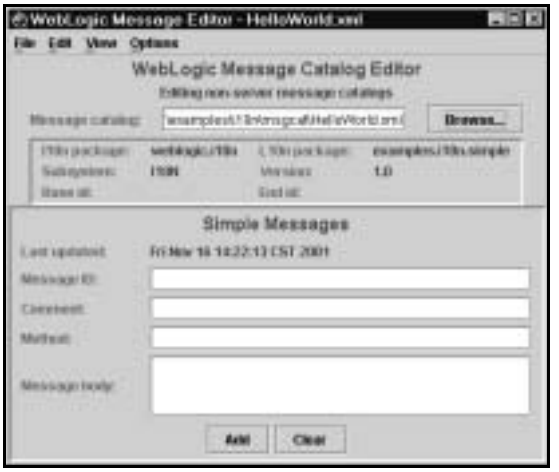
カタログが見つかると、そのカタログの**パッケージ**、**サブシステム**、**バージョン**、および**開始 ID** と**終了 ID** (ある場合) が表示され、そのカタログが、他のすべてのアクションが実行されるカタログ コンテキストになります。この時点で、カタログでは、新しいメッセージの入力、既存のメッセージの編集、メッセージの検索、またはすべてのメッセージの表示を行うことができます。

[**Message Catalog**] フィールドで、ログ メッセージ カタログが選択された場合、図 3-3に示されているような**ログ メッセージ**用の WebLogic メッセージ エディタ ウィンドウが表示されます。

図 3-3 ログ メッセージ用 WebLogic メッセージ エディタ

[**Message Catalog**] フィールドで、シンプル メッセージ カタログが選択された場合、図 3-4 に示されているような**シンプル メッセージ**用の WebLogic メッセージ エディタ ウィンドウが表示されます。

図 3-4 シンプル メッセージ用 WebLogic メッセージ エディタ



新規カタログの作成

新規カタログを作成するには、次の手順に従います。

1. WebLogic メッセージ エディタ ウィンドウのメイン メニュー バーから [**File**] メニューを選択します。
2. [**New Catalog**] を選択します。

   図 3-5 に示されているように、[Create new catalog] ダイアログが表示されます。

図 3-5 [Create new catalog] ダイアログ

3. [**Message Catalog**] フィールドに、新規カタログ (拡張子 `xml` を含むファイル) の絶対パス名と名前を入力します。または、[**Browse**] をクリックして、WebLogic カタログ ディレクトリ、`msgcat` に移動します。

4. [**Catalog type**] ドロップダウン リストを使用して、作成するカタログが、**ログ メッセージ (Log message)** カタログ、**シンプル テキスト (Simple text)** メッセージ カタログのいずれかを指定します。

   ログ メッセージ カタログを選択した場合は、[**Base ID**] および [**End ID**] フィールドが表示されます。この 2 つのフィールドは、カタログ内のメッセージに使用する ID 番号の範囲を指定するものです。シンプル テキスト メッセージ カタログを選択した場合は、これらのフィールドは表示されません。

5. 生成された Logger クラスを入れたいパッケージの名前を [**I18n package**] フィールドに入力します。デフォルト値は `weblogic.i18n` です。Logger クラスをアプリケーションと共に別のパッケージに入れたい場合は、ここでそのパッケージ名を指定します。

6. カタログ データを入れたいパッケージの名前を [**L10n package**] フィールドに入力します。デフォルト値は `weblogic.i18n` です。カタログ データをアプリケーションと共に別のパッケージに入れたい場合は、ここでそのパッケージ名を指定します。

7. [**Subsystem**] フィールドに、メッセージをログに記録したシステムの部分を示す名前を入力します。この名前がメッセージと共にログに書き込まれます。アプリケーションの場合、通常、[**Subsystem**] フィールドにはそのアプリケーション名が入ります。

8. [**Create Catalog**] をクリックします。

[Create new catalog] ダイアログが閉じ、今作成したカタログが、メッセージ エディタ のメイン ウィンドウにコンテキスト カタログとして表示されます。

カタログへのメッセージの追加

以下の節では、メッセージ エディタを使用してカタログにメッセージを追加する方法について説明します。

- 新規ログ メッセージの入力
- 新規シンプル テキスト メッセージの入力

新規ログ メッセージの入力

ログ カタログに新規メッセージを入力するには、次の手順に従います。

1. 図 3-6 に示す WebLogic メッセージ エディタのメイン ダイアログで、[**Message Catalog**] フィールドに絶対パス名を入力するか、[**Browse**] をクリックして既存のカタログに移動します。

図 3-6　ログ メッセージ

2. 英数字のユニークな**メッセージ ID (Message ID)** を入力するか、または [**Get next ID**] ボタンをクリックしてコンテキスト カタログ内でユニークな次の ID を取得します。

3. 括弧と引数も含めて、ログ メッセージに使用する適切な**メソッド (Method)** を入力します。
   例：

   ```
   logNoAuthorization(String arg0, java.util.Date arg1,
      int arg2) logNote()
   ```

4. リストから**重要度 (Severity)** を選択します。

5. **メッセージ本文 (Message body)**、**メッセージ詳細 (Message detail)**、**考えられる原因 (Possible cause)**、および**アクション (Action)** に対応するテキストを入力します。パラメータは {*n*} で示します。
   例：

```
User {0} tried to access this on {1} but has no authority to do
so.{2} lashes with a keyboard with coke spilled on it.
```

6. チェックマーク ボックスをクリックして、[**Display stacktrace**] オプションのオン、オフを切り替えます。このオプションは、Logger メソッドが例外を引数として取る場合に、メッセージと共にスタックトレースを出力するときに使用します。
7. [**追加**] をクリックします。

   メッセージが追加されると、カタログ全体がただちにディスクに書き込まれます。

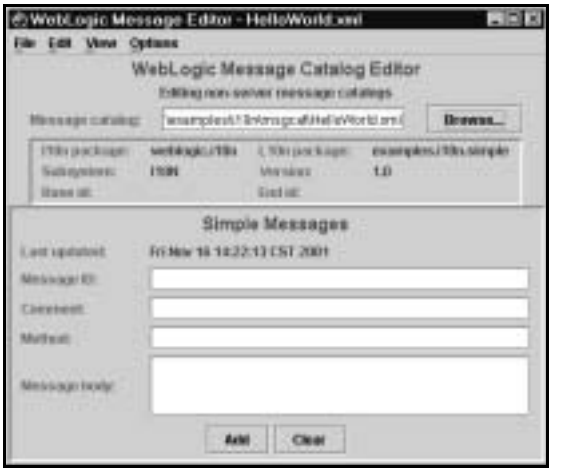
新規シンプル テキスト メッセージの入力

シンプル メッセージ カタログにシンプル テキスト メッセージを入力するには、次の手順に従います。

1. WebLogic メッセージ エディタのメイン ダイアログで、[**Message Catalog**] フィールドに絶対パス名を入力するか、[**Browse**] をクリックして既存のカタログに移動します。

   図 3-7 に示されているようなシンプル メッセージ用の WebLogic メッセージ エディタのメイン ウィンドウが表示されます。

図 3-7　シンプル メッセージ用ウィンドウ



2. ユニークな英数字の**メッセージ ID (Message ID)** を入力します。
3. 必要に応じて、**コメント (Comment)** を入力します。
4. 括弧と引数も含めて、シンプル メッセージに使用する適切な**メソッド (Method)** を入力します。
5. **メッセージ本文 (Message body)** を入力します。
6. [**追加**] をクリックします。

   メッセージが追加されると、カタログ全体がただちにディスクに書き込まれます。

メッセージの検索

以下の節では、メッセージ エディタをメッセージ検索に使用する方法について説明します。

- ログ メッセージの検索
- シンプル テキスト メッセージの検索

ログ メッセージの検索

ログ メッセージを検索するには、次の手順に従います。

1. コンテキスト カタログがログ メッセージ カタログであること、図 3-3 に示されているように WebLogic メッセージ エディタの**ログ メッセージ**のメイン ウィンドウが表示されていることを確認します。
2. メイン メニュー バーから [**Edit**] を選択します。
3. [**Search**] を選択して、図 3-8 に示すような [Search for Log Message] ダイアログを表示します。

図 3-8 ログ メッセージの検索

4. **メッセージ ID (Message ID)** および**メソッド (Method)** 名を入力します。
5. [**Message text search**] に、求めるメッセージの検索データを必要なだけ入力します。テキストの検索では、テキスト フィールドに入力されたすべての内容について部分一致検索が行われます。
6. [**Find first**] または [**Find next**] をクリックします。

   メッセージの検索は、フィールドの情報をつなぎ合わせて実行されます。一致するメッセージが見つかると、図 3-1 に示されているようにそのメッセージがメッセージ エディタのメイン ウィンドウに表示されます。

シンプル テキスト メッセージの検索

シンプル テキスト メッセージを検索するには、次の手順に従います。

1. コンテキスト カタログがシンプル テキスト メッセージ カタログであること、図 3-4 に示されているように WebLogic メッセージ エディタの**シンプル メッセージ**のメイン ウィンドウが表示されていることを確認します。
2. メイン メニュー バーから [**Edit**] を選択します。
3. [**Search**] を選択して、図 3-9 に示すような [Search for Simple Message] ダイアログを表示します。

図 3-9 シンプル メッセージの検索

4. **メッセージ ID (Message ID)** を入力します。
5. [**Message text search**] に、求めるメッセージの検索データを必要なだけ入力します。テキストの検索では、テキスト フィールドに入力されたすべての内容について部分一致検索が行われます。
6. [**Find first**] または [**Find next**] をクリックします。

   メッセージの検索は、フィールドの情報をつなぎ合わせて実行されます。一致するメッセージが見つかると、図 3-4 に示されているようにそのメッセージがメッセージ エディタのメイン ウィンドウに表示されます。

Message Viewer の使い方

WebLogic メッセージ エディタには、Message Viewer が付属しており、このビューアを使えば、1 つのカタログ内の全メッセージおよび複数のカタログ内の全メッセージを表示すること、さらに任意のメッセージを選択して編集することができます。

以下の節では、Message Viewer を使用して、メッセージを表示する方法、およびメッセージを選択して編集する方法を説明します。

- 1 つのカタログ内の全メッセージの表示
- 複数のカタログ内の全メッセージの表示
- Message Viewer から編集対象のメッセージを選択する

1 つのカタログ内の全メッセージの表示

1 つのカタログ内のすべてのメッセージを表示する手順は次のとおりです。

1. WebLogic メッセージ エディタを開きます。WebLogic メッセージ エディタのメイン ウィンドウに、最後に表示したメッセージのカタログが、現在のコンテキスト カタログとして表示されます。
2. メニュー バーから **[View]** を選択します。現在のコンテキスト カタログのすべてのメッセージが、図 3-10 に示されているように Message Viewer ウィンドウに表形式で表示されます。Message Viewer は、メッセージ エディタとは別のウィンドウに表示されます。メッセージ エディタは開いたままとなります。

図 3-10 Message Viewer

複数のカタログ内の全メッセージの表示

現在のコンテキスト カタログからメッセージを表示し、次に WebLogic メッセージ エディタのメイン ウィンドウで [**Browse**] をクリックして新しいカタログに移動してコンテキストを変更すると、前のカタログの表示は元の画面に残り、新しいカタログの内容は別の Message Viewer ウィンドウに表示されます。この操作を繰り返すと、必要な (または画面に表示が収まる限りの) 数のカタログを同時に表示できます。カタログごとに別々の Message Viewer ウィンドウが表示されます。新規のカタログの参照方法については、3-4 ページの「既存のカタログの参照」を参照してください。

Message Viewer から編集対象のメッセージを選択する

Message Viewer を使用して、メッセージ リストを表示後、ビューアの行に表示された任意のメッセージをクリックして選択することができます。選択したメッセージのカタログがコンテキスト カタログとなり、そのメッセージがメッセージ エディタのメイン ウィンドウに表示されます。

図 3-11 メッセージ 600002 に対する Message Viewer とメッセージ エディタ

既存のメッセージの編集

既存のメッセージを編集するには、次の手順に従います。

1. 「ログ メッセージの検索」および「シンプル テキスト メッセージの検索」に記載されているように [Search] ダイアログを使用して検索するか、「Message Viewer から編集対象のメッセージを選択する」に示されているように Message Viewer で行をクリックして、編集するメッセージを指定します。
2. メッセージ エディタのメイン ウィンドウでフィールドを編集します。
3. [**Update**] をクリックします。

メッセージが更新され、カタログ全体がただちにディスクに書き込まれます。

4 BEA WebLogic Server インターナショナライゼーション ユーティリティの使い方

以下の節では、インターナショナライゼーションおよびローカライゼーションに使用される WebLogic Server ユーティリティに関する情報を記述します。

- WebLogic Server インターナショナライゼーション ユーティリティ
- WebLogic Server のインターナショナライゼーションおよびローカライゼーション インタフェース
- 18ngen ユーティリティ
- l10ngen ユーティリティ
- CatInfo ユーティリティ

WebLogic Server インターナショナライゼーション ユーティリティ

WebLogic Server では、以下の 3 つのインターナショナライゼーション ユーティリティが用意されています。

- 18ngen ユーティリティ — メッセージ カタログのパーサ。このユーティリティは、ログ メッセージ内のテキストのローカライズに使用されるクラスの生成に使用します。
- l10ngen ユーティリティ — ロケール固有のメッセージ カタログのパーサ。このユーティリティは、ロケール固有のカタログの処理に使用します。

- CatInfo ユーティリティ — 組み込まれているログ メッセージのリストを生成するユーティリティ。このユーティリティは、組み込まれているログ メッセージのリストの生成に使用します。

注意： カタログ定義のテキストでは読みやすいように書式設定文字 (行末文字など) が使用される場合がありますが、それらの文字はパーサでは維持されません。テキスト データは、1 行の文字列に標準化されます。先頭と末尾のホワイト スペースはすべて削除されます。埋め込まれている行末文字は、単語の区切りを維持するために必要に応じてスペースに置き換えられます。タブはそのまま残されます。

WebLogic Server のインターナショナライゼーションおよびローカライゼーション インタフェース

`i18ngen` ユーティリティは、メッセージ カタログを検証し、ローカライズされたメッセージの生成に必要な実行時クラスを作成します。`l10ngen` ユーティリティは、ロケール固有のカタログを検証し、カタログで定義されたさまざまな異なるロケールで使用される追加のプロパティ ファイルを作成します。

WebLogic Server で実行しているシンプル テキストベースのユーティリティは、それらのユーティリティでテキスト データのアクセスに `Localizer` を使用するように指定することにより、インターナショナライズできます。`i18ngen` ユーティリティから生成された `Logger` および `TextFormatter` クラスをアプリケーションに組み込みます。`i18ngen` ユーティリティの詳細については、4-4 ページの「18ngen ユーティリティ」を参照してください。

生成された `Logger` クラスは、英語のテキストをログに書き込む従来の方法に代わるログ記録の手段として使用されます。たとえば、`i18ngen` では、カタログ `xyz.xml` に対して適切なパッケージにクラス `xyzLogger` が生成されます。

もう 1 つ例を挙げると、たとえば `MyUtilLog.xml` カタログが使用される場合は、クラス `programs.utils.MyUtilLogger.class` が生成されます。カタログで定義されているログ メッセージごとに、このクラスでは `method` 属性で定義されているとおりに静的なパブリック メソッドが格納されます。

TextFormatter クラスは、シンプル メッセージ カタログごとに生成されます。TextFormatter クラスには、カタログのローカライズおよびフォーマットされたテキストにアクセスするために使用する静的なメソッドがあります。このクラスは、メッセージ本文、プレースホルダ、および MessageFormat とのインタフェースを処理するコンビニエンス クラスです。フォーマット用メソッドは、各メッセージ定義の method 属性により指定します。たとえば、あるカタログのメッセージ定義に属性 method=getErrorNumber(int err) が指定されている場合、コード リスト 4-1 に示されている TextFormatter クラスが生成されます。

コード リスト 4-1　TextFormatter クラスの例

```
package my.text;
public class xyzTextFormatter
{
   . . .
   public String getErrorNumber(int err)
   {
   . . .
   }
}
```

コード リスト 4-2 は、getErrorNumber メソッドのコードでの使用例を示しています。

コード リスト 4-2　getErrorNumber メソッドの例

```
import my.text.xyzTextFormatter
. . .

xyzTextFormatter xyzL10n = new xyzTextFormatter();
System.out.println(xyzL10n.getErrorNumber(someVal));
```

この場合、someVal 引数が適切に挿入された状態で現在のロケールのメッセージテキストが出力されます。

18ngen ユーティリティ

i18ngen ユーティリティでは、メッセージ カタログ (XML ファイル) が解析され、ログ メッセージのテキストをローカライズするための Logger クラスおよび TextFormatter クラスが生成されます。最上位のメッセージ カタログは、Java クラスおよびプロパティ ファイルにコンパイルされます。i18ngen ユーティリティでは、i18n_user.properties ファイルの作成または更新も行われます。エラーや警告などのメッセージはすべて stderr に送信されます。

構文

```
java weblogic.i18ngen [options] files
```

注意 : ユーティリティはどのディレクトリからでも実行できますが、ファイルがコマンド ラインで指定される場合、ファイル パスはカレント ディレクトリに対する相対パスになります。

オプション

オプション	定義
-d *targetdirectory*	生成された Java ソース ファイルの宛先となるルート ディレクトリを指定する。ユーザ カタログ プロパティは、指定された対象ディレクトリの相対パスにある i18n_user.properties に配置される。ファイルは、メッセージ カタログの i18n_package 値および l10n_package 値に基づいて適切なディレクトリに配置される。デフォルトの対象ディレクトリはカレント ディレクトリとなる。
-n	解析し、有効性を検証するが、クラスは生成しない。
-keepgenerated	生成された Java ソースを維持する。
-ignore	エラーを無視する。

`-i18n`	インターナショナライズされたメッセージのロギングをサポートするインターナショナライザ ソース (`*Logger.java` など) を作成する。
`-l10n`	各メッセージ カタログで定義されている各メッセージにアクセスできるようにするローカライザ ソース (プロパティ リソースのまとまり) を作成する。それらのプロパティは、メッセージをローカライズするためにローカライゼーション ユーティリティによって使用される。
`-compile`	現在の CLASSPATH を使用して生成された Java ファイルをコンパイルする。生成されたクラスは、-d オプションで識別されるディレクトリに配置される。コンパイルの途中でエラーが検出されると、クラス ファイルやプロパティ ファイルが作成されず、i18ngen は異常な終了ステータスで終了する。
`-nobuild`	解析と有効性の検証のみを実行する。
`files`	このファイル リストのファイルとディレクトリを処理する。ディレクトリが指定されている場合は、そのディレクトリのすべての XML ファイルが処理される。すべてのファイルの名前には、XML のサフィックスが付いている必要がある。すべてのファイルは `msgcat.dtd` 構文に準拠していなければならない。`i18ngen` は、実際に生成されたファイルに対して、完全修飾名リスト (Java ソース) を `stdout` ログに出力する。

l10ngen ユーティリティ

l10ngen ユーティリティでは、最上位カタログの下位に位置するディレクトリ内のロケール固有のカタログが処理されます。

構文

```
java -classpath <l10n_Classpath> weblogic.i18ntools.l10ngen
[options] filelist
```

ここで、<l10n_Classpath> には <WebLogic Home>/lib/weblogic.jar が含まれている必要があります。

注意： ユーティリティはどのディレクトリからでも実行できますが、ファイルがコマンド ラインで指定される場合、ファイル パスはカレント ディレクトリに対する相対パスになります。

オプション

オプション	定義
-d *targetdirectory*	プロパティを配置するディレクトリ。デフォルトはカレント ディレクトリ。
-language *code*	言語コード。デフォルト値は all。
-country *code*	国コード。デフォルト値は all。
-variant *code*	バリアント コード。デフォルト値は all。
-*filelist*	処理するメッセージ カタログのディレクトリとファイル。カレント ディレクトリを基準に指定する。ロケール固有のディレクトリではなく最上位のディレクトリを指定する。

CatInfo ユーティリティ

このユーティリティでは、組み込まれているログ メッセージのリストが生成されます。デフォルトの `CatInfo` では、現在組み込まれているすべてのログ メッセージの ID とメッセージ本文が順番に表示されます。

構文

```
java weblogic.i18ntools.CatInfo [options]
```

注意： ユーティリティはどのディレクトリからでも実行できますが、ファイルがコマンド ラインで指定される場合、ファイル パスはカレント ディレクトリに対する相対パスになります。

オプション

注意： すべてのオプションは 1 文字に短縮できます。

オプション	定義
`-id` *nnnnnn*	*nnnnnn* はメッセージ ID を表す。 `-id` オプションは特定のメッセージを指定するために使用する。
`-subsystem` *identifier*	サブシステムの識別子。`-subsystem` オプションでは指定されたサブシステムに一致するメッセージだけが出力される。
`-detail`	詳細なリストを要求する。`-detail` オプションではバージョン、重要度、サブシステム、メッセージの詳細、原因、およびアクションといった情報も要求される。
`-help`	ヘルプ情報を提供する。

メッセージの詳細リストをファイルにエクスポートする場合は、次のコマンドを使用してください。

```
java weblogic.i18ntools.CatInfo -detail > Errors.txt
```

A BEA WebLogic Server 用の Localizer クラスのリファレンス

以下の節では、Localizer クラスのリファレンス情報を提供します。

- Localizer クラスの概要
- Localizer メソッド
- Localizer のルックアップ クラス

注意： この Localizer クラス メソッドの情報は、上級ユーザ向けのリファレンスです。通常、ユーザがこのインタフェースを直接使用する必要はありません。

Localizer クラスの概要

Localizer は、出力用のテキストをローカライズするためにアプリケーションやサーバ コードによって使用されるクラスです。i18ngen ユーティリティでは、メッセージ カタログの内容に基づいて **Localizer** クラスが作成されます。

Localizer クラスは、カタログ ファイルごとに 1 つ生成されます。クラスの名前は、カタログ名 (.xml 拡張子はユーティリティによって削除される) の後に **LogLocalizer** を付けたものです。カタログ ejb.xml の **Localizer** クラスは、ejb**LogLocalizer** です。

Localizer メソッド

Localizer は、java.util.ListResourceBundle クラスを拡張したものです。Localizer でのローカライゼーション データのアクセスを容易にするために、4 つのメソッドが追加されています。これらのメソッドは、以下の表 A-1 で説明されています。

表 A-1 ローカライゼーション データのアクセスに使用されるメソッド

メソッド	説明
public Object getObject(String key, String id)	メッセージ id の key 要素に対応するローカライゼーション テキストを返す。
public Object getObject(String key, int id)	メッセージ id の key 要素に対応するローカライゼーション テキストを返す。
public String getString(String key, String id)	メッセージ id の key 要素に対応するローカライゼーション テキストを返す。
public String getString(String key, int id)	メッセージ id の key 要素に対応するローカライゼーション テキストを返す。

ローカライゼーション データ アクセスに使用される各メソッドには key 引数があります。次に挙げるのは、key 引数として識別される値です。

- Localizer.SEVERITY
- Localizer.MESSAGE_ID
- Localizer.MESSAGE_BODY
- Localizer.MESSAGE_DETAIL

- Localizer.CAUSE
- Localizer.ACTION

Localizer.SEVERITY キーの場合を除き、Localizer によって返されるローカライゼーション データは整数型のオブジェクトを返す String オブジェクトです。

返される重要度 (severity) の値は、以下のとおりです。

- weblogic.logging.severities.EMERGENCY
- weblogic.logging.severities.ALERT
- weblogic.logging.severities.CRITICAL
- weblogic.logging.severities.ERROR
- weblogic.logging.severities.WARNING
- weblogic.logging.severities.NOTICE
- weblogic.logging.severities.INFO
- weblogic.logging.severities.DEBUG

返される特定の文字列は、メッセージ カタログで定義されます。

get*() メソッドの key 引数は、定義のどの要素を返すのかを識別します。受け入れられる値は、Localizer クラスの定義で定義されます。返されるテキストは、java.text.MessageFormat.format() によってさらに拡張できます。message body、detail、cause、および action の各要素は、すべてローカライズできます。他の要素 (message ID、severity、および subsystem) はローカライズ可能ではなく、MessageFormat で処理を加える必要もありません。

Localizer のルックアップ クラス

メッセージに対して正しい Localizer を取得するには、L10nLookup クラスを使用します。L10nLookup は、システムの起動時に次のプロパティ ファイルからロードされる Property クラスの拡張です。

\weblogic\i18n\i18n.properties

このプロパティ ファイルは、i18ngen によって作成されます。ルックアップ ファイルのプロパティのフォーマットは次のとおりです。

nnnnnn=subsystem:*Localizer class*

この行の引数は次のように定義されています。

- *nnnnnn* はメッセージ ID を表します。
- *subsystem* は関連するサブシステムを表します。
- *Localizer class* は生成された Localizer クラスの名前を表します。

たとえばメッセージ 001234 は、次のルックアップ ファイルのプロパティに基づいて、weblogic.i18n.ejbLogLocalizer クラスからの EJB サブシステム メッセージ ID として識別されます。

```
001234=EJB:weblogic.i18n.ejbLogLocalizer
```

B BEA WebLogic Server 用の Logger クラスのリファレンス

以下の節では、Logger クラスのリファレンス情報を提供します。

- Logger クラスの概要
- 生成された Logger クラスの例

注意： この Logger クラスの情報は、通常使用するメソッドのリファレンスとして提供しているものです。通常、ユーザがこのインタフェースを直接使用する必要はありません。

Logger クラスの概要

i18ngen によって生成されるクラスは、Loggers と呼ばれています。Logger クラスは、WebLogic Server エラー ログのインタフェースを提供します。カタログ Xyz.xml の場合、Logger クラス **XyzLogger** が生成されます。

Logger クラスは、カタログに定義されたすべてのメッセージを WebLogic Server ログに記録するメソッドを提供します。組み込まれているメソッドは、関連するカタログで定義されたものと同じです。カタログで loggables 属性が true に指定されていると、メッセージごとに Loggable メソッドが併せて生成されます。

生成された Logger クラスの例

コード リスト B-1 は、メッセージ定義が 1 つあるカタログの例を示しています。

リスト B-1 メッセージ カタログの例

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls600/msgcat.dtd">
<message_catalog
   i18n_package="examples.i18n.logging"
   l10n_package="examples.i18n.logging"
   subsystem="I18N"
   version="1.0"
   baseid="600000"
   endid="610000"
   loggables="true"
   >
   <logmessage
     messageid="600000"
     method="logEntry()"
     severity="info"
     >
     <messagebody>Starting I18nLog example...</messagebody>
     <messagedetail></messagedetail>
     <cause></cause>
     <action></action>
   </logmessage>
   <logmessage
     messageid="600001"
     method="testArgs(String name,int cnt)"
     severity="debug"
     >
     <messagebody>Class {0} started with {1,number}
arguments.</messagebody>
     <messagedetail></messagedetail>
     <cause></cause>
     <action></action>
   </logmessage>
   <logmessage
     messageid="600002"
     method="logTrace(Throwable t)"
     severity="error"
     stacktrace="true"
     >
    <messagebody>This message is followed by a trace</messagebody>
     <messagedetail></messagedetail>
     <cause></cause>
     <action></action>
   </logmessage>
   <logmessage
     messageid="600003"
     method="logNoTrace(Throwable t)"
     severity="warning"
     stacktrace="false"
     >
     <messagebody>This message is not followed by a trace, but we
can insert its text : {0}</messagebody>
     <messagedetail></messagedetail>
```

```
      <cause></cause>
      <action></action>
   </logmessage>
   <logmessage
      messageid="600004"
      method="getId()"
      severity="info"
      >
      <messagebody>This message's id will be in the next
message</messagebody>
      <messagedetail>A message can contain additional detailed
information.</messagedetail>
      <cause>This message is displayed on purpose</cause>
      <action>Nothing to do, the example is working</action>
   </logmessage>
   <logmessage
      messageid="600005"
      method="showId(String id)"
      severity="info"
      >
      <messagebody>The previous message logged had message id
{0}</messagebody>
      <messagedetail></messagedetail>
      <cause></cause>
      <action></action>
   </logmessage>
</message_catalog>
```

コード リスト B-2 は、これに対応する i18ngen が生成した Java ソースを示しています。

リスト B-2　生成された Logger クラスの例

```
package examples.i18n.logging;

import weblogic.logging.MessageLogger;
import weblogic.logging.Loggable;
import java.util.MissingResourceException;

/**
 * Copyright (c) 2001 by BEA Systems, Inc. All Rights Reserved.
 * @exclude
 */
public class I18nLogLogger
{
  /**
   * Starting I18nLog example...
   * @exclude
```

```
   *
   * messageid:  600000
   * severity:   info
   */
  public static String logEntry()  {
    Object [] args = {  };
    MessageLogger.log(
       "600000",
       args,
       "examples.i18n.logging.I18nLogLogLocalizer");
    return "600000";
  }
  public static Loggable logEntryLoggable() throws MissingResourceException {
    Object[] args = {  };
    return new Loggable("600000", args);
  }
  /**
   * Class {0} started with {1,number} arguments.
   * @exclude
   *
   * messageid:  600001
   * severity:   debug
   */
  public static String testArgs(String arg0, int arg1)  {
    Object [] args = { arg0, new Integer(arg1) };
    MessageLogger.log(
       "600001",
       args,
       "examples.i18n.logging.I18nLogLogLocalizer");
    return "600001";
  }
  public static Loggable testArgsLoggable(String arg0, int arg1) throws
MissingResourceException {
    Object[] args = { arg0, new Integer(arg1) };
    return new Loggable("600001", args);
  }
  /**
   * This message is followed by a trace
   * @exclude
   *
   * messageid:  600002
   * severity:   error
   */
  public static String logTrace(Throwable arg0)  {
    Object [] args = { arg0 };
    MessageLogger.log(
       "600002",
       args,
       "examples.i18n.logging.I18nLogLogLocalizer");
```

```
    return "600002";
  }
  public static Loggable logTraceLoggable(Throwable arg0) throws
MissingResourceException {
    Object[] args = { arg0 };
    return new Loggable("600002", args);
  }
  /**
   * This message is not followed by a trace, but we can insert its text : {0}
   * @exclude
   *
   * messageid:  600003
   * severity:   warning
   */
  public static String logNoTrace(Throwable arg0)  {
    Object [] args = { arg0 };
    MessageLogger.log(
       "600003",
       args,
       "examples.i18n.logging.I18nLogLogLocalizer");
    return "600003";
  }
  public static Loggable logNoTraceLoggable(Throwable arg0) throws
MissingResourceException {
    Object[] args = { arg0 };
    return new Loggable("600003", args);
  }
  /**


   * This message's id will be in the next message
   * @exclude
   *
   * messageid:  600004
   * severity:   info
   */
  public static String getId()  {
    Object [] args = {  };
    MessageLogger.log(
       "600004",
       args,
       "examples.i18n.logging.I18nLogLogLocalizer");
    return "600004";
  }
  public static Loggable getIdLoggable() throws MissingResourceException {
    Object[] args = {  };
    return new Loggable("600004", args);
  }
  /**
```

```
   * The previous message logged had message id {0}
   * @exclude
   *
   * messageid:  600005
   * severity:   info
   */
  public static String showId(String arg0)  {
    Object [] args = { arg0 };
    MessageLogger.log(
       "600005",
       args,
       "examples.i18n.logging.I18nLogLogLocalizer");
    return "600005";
  }
  public static Loggable showIdLoggable(String arg0) throws
MissingResourceException {
    Object[] args = { arg0 };
    return new Loggable("600005", args);
  }

}
```

コード リスト B-3 は、i18nLog を使用するサンプル アプリケーションを示しています。

リスト B-3 i18nLog を使用するアプリケーション例

```
package examples.i18n.logging;

import java.util.Locale;

import weblogic.i18n.Localizer;
import weblogic.i18ntools.L10nLookup;
import weblogic.logging.Loggable;

/**
 * @author Copyright (c) 2000 by BEA Systems, Inc. All Rights Reserved.
 */

/**
 * この例では、インターナショナライズ (I18n) ロギング インタフェースの使用法を示す
 * <p>
 * 使い方 : java examples.i18n.logging.I18nLog
 * <p>
```

```
 * ビルド プロシージャ : run bld.sh (UNIX) または bld.cmd (NT)。このスクリプトにより、
 * I18nLog.xml カタログが処理され、ロギング クラス、
 * <tt>examples.i18n.logging.I18nLogLogger</tt> が生成される。このクラスには、WLS エラー ログに対してメッセージを書き込む静的メソッドが組み込まれている。このメソッドと引数は、
 * I18nLog.xml カタログに定義されています。この例では、単純なメッセージ カタログ、
 * I18nSimple.xml も使用している
 */


public  class I18nLog {

    public I18nLog() {}

    public static void main(String[] argv) {
      /**
       * この呼び出しでは、info メッセージがログに書き込まれるだけである。このメソッドには、
       * 引数は定義されていない
       *
       * ここでは、メソッドの Loggable 形式の使用方法も示している
       */

      Loggable ll = I18nLogLogger.logEntryLoggable();
      ll.log();
      System.out.println(ll.getMessage());

      /**
       * ここに示すのは、さまざまな引数を含むメッセージのサンプル
       * である
       */
      I18nLogLogger.testArgs(I18nLog.class.getName(),argv.length);
      /**
       * Throwable が渡された場合は、デフォルトにより、メソッドと共に
       * スタックトレースがログに書き込まれる
       */
      Throwable t = new Throwable("Test with stack trace");
      I18nLogLogger.logTrace(t);
      /**
       * スタック トレースをログ記録しないようにメッセージを定義することもできる
       */
      I18nLogLogger.logNoTrace(t);
      /**
       * logger メソッドは、これらのメッセージのログだけではなく他の処理も
       * 行うアプリケーションの messageid を返す
       */
      String messageId = I18nLogLogger.getId();
      I18nLogLogger.showId(messageId);
      /**
```

```
     * メッセージ ID は、メッセージのさまざまな属性の入手に使用でき
     * る。L10nLookup オブジェクトにより、Localizer クラスを通じてカタログにアクセス
     * できる。Localizer により、個々のメッセージにアクセス
     * できる。各ログ メッセージ カタログには、一般メッセージ情報用と
     * 詳細属性用の 2 つの Localizer が入っている
     *
     * 基本 Localizer によって、以下のカタログ情報にアクセスする
     *      Version
     *      L10n Package —カタログ データ用パッケージ
     *      I18n Package — Logger メソッド用パッケージ
     *      Subsystem —カタログ サブシステム
     * また、メッセージごとに以下の情報を提供する
     *      Severity: debug (128)、info (64)、warning (32)、error (8)
     *      Message Body —メッセージ テキスト
     *      Stack option —スタック トレースをログに記録するかどうかの選択
     *
     * まず、L10nLookup プロパティを開き、プロパティを使用して、
     * メッセージに対する Localizer の内容を入手する
     */
    L10nLookup l10n = L10nLookup.getL10n();
    /**
     * 以下のコードは、基本 Localizer (arg 3 = false) を返す
     */
    Localizer lcl = l10n.getLocalizer(messageId,Locale.getDefault(),false);
    /**
     * 以下のコードは、詳細 Localizer (arg 3 = true) を返す
     */


    Localizer lclDetail = l10n.getLocalizer(messageId,Locale.getDefault(),true);
    /**
     * このアプリケーションのシンプル メッセージ カタログは、ログ メッセージ
     * カタログ情報の表示に使用する
     */
    I18nSimpleTextFormatter fmt = new I18nSimpleTextFormatter();
    System.out.println(fmt.version(messageId,lcl.getVersion()));
    System.out.println(fmt.l10nPackage(messageId,lcl.getL10nPackage()));
    System.out.println(fmt.i18nPackage(messageId,lcl.getI18nPackage()));
    System.out.println(fmt.subsystem(messageId,lcl.getSubSystem()));
    System.out.println(fmt.severity(messageId,lcl.getSeverity(messageId)));
    System.out.println(fmt.body(messageId,lcl.getBody(messageId)));
    System.out.println(fmt.stack(messageId,lcl.getStackTrace(messageId)));
    /**
     * 次は詳細情報
     */
    System.out.println(fmt.detail(messageId,lclDetail.getDetail(messageId)));
    System.out.println(fmt.cause(messageId,lclDetail.getCause(messageId)));
```

```
      System.out.println(fmt.action(messageId,lclDetail.getAction(messageId)));

    }
}
```

コード リスト B-4 は、i18ngen が生成した、これに対応する Java ソースを示しています。

リスト B-4　生成された Logger クラスの例

C BEA WebLogic Server 用の Loggable オブジェクトのリファレンス

以下の節では、Loggable オブジェクトのリファレンス情報を提供します。

- Loggable オブジェクトの概要
- Loggable オブジェクトの使用例

Loggable オブジェクトの概要

デフォルトでは、すべてのログ メッセージ カタログは、メッセージの WebLogic Server ログへの書き込みに使用されるメソッドを含む Logger クラスを作成します。Logger クラスには、メッセージをログに記録せずに、Loggable Objects を返すメソッドを任意選択で組み込むことができます。Loggable オブジェクトは、ログ メッセージを生成するが、実際のログへの書き込みは後から行いたい場合に便利です。また、例外送出など、他の目的にメッセージ テキストを使用する場合にも使用できます。

Loggable オブジェクトの使用例

Loggable オブジェクトを返すメソッドを持つ Logger クラスを作成するには、メッセージ カタログに loggables 属性を設定する必要があります。

たとえば、コード リスト C-1 に示す test.xml カタログを想定します。

リスト C-1 test.xml メッセージ カタログ

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls600/msgcat.dtd">
<message_catalog
   subsystem="Examples"
   version="1.0"
   baseid="500000"
   endid="500001"
   loggables="true"
   >
   <logmessage
      messageid="500000"
      severity="error"
      method="logIOError(Throwable t)"
      >
      <messagebody>
         IO failure detected.
      </messagebody>
      <messagedetail>
      </messagedetail>
      <cause>
      </cause>
      <action>
      </action>
   </logmessage>
</message_catalog>
```

このカタログを i18ngen ユーティリティで実行すると、このカタログに対して次の 2 つのメソッドを持つ Logger クラスが作成されます。

- logIOError (Throwable)―メッセージをログに記録する
- logIOErrorLoggable (Throwable)―Loggable オブジェクトを返す

Loggable クラスの使用例は、以下の コード リスト C-2 に示されています。

リスト C-2 Loggable クラスの使用例

```
package test;
import weblogic.logging.Loggable;
import weblogic.i18n.testLogger;

...
try {
  // 何らかの入出力
```

```
} catch (IOException ioe) {
  Loggable l = testLogger.logIOErrorLoggable(ioe);
  l.log();  // エラーをログに書き込む
  throw new Exception(l.getMessage());// ログされたものと同じテキストで
        新規の例外を送出する
}
```

D BEA WebLogic Server 用の TextFormatter クラスのリファレンス

以下の節では、TextFormatter クラスのリファレンス情報を提供します。

- TextFormatter クラスの概要
- TextFormatter クラスを使用したアプリケーションの例

注意： この TextFormatter クラスの情報は、通常使用するメソッドのリファレンスとして提供しているものです。通常、ユーザがこのインタフェースを直接使用する必要はありません。

TextFormatter クラスの概要

TextFormatter クラスは、i18ngen によってシンプル メッセージ カタログから生成されます。このクラスは、実行時にメッセージ テキストのローカライズ版の生成に使用されるメソッドを提供します。次の節では、アプリケーションとそのシンプル メッセージ カタログ、およびそのカタログに対して生成された TextFormatter クラスの例を示します。

TextFormatter クラスを使用したアプリケーションの例

コード リスト 4-3 は、「Hello World」アプリケーションに対するシンプル メッセージ カタログの例を示しています。

コード リスト 4-3 シンプル メッセージ カタログの例

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls600/msgcat.dtd">
<message_catalog

   l10n_package="examples.i18n.simple"

   subsystem="I18N"

   version="1.0"

   >

   <message
      messageid="HELLO_WORLD"
      datelastchanged="967575717875"
      method="helloWorld()"
      >
      <messagebody>
         Hello World!
      </messagebody>
   </message>

   <!--  -->
   <message
      messageid="HELLO_AGAIN"
      datelastchanged="967575717804"
      method="helloAgain()"
      >
      <messagebody>
         Hello again
      </messagebody>
   </message>

   <!--  -->
   <message
      messageid="NTH_HELLO"
      datelastchanged="967575770971"
```

```
      method="nthHello(int count)"
      >
      <messagebody>
         This is hello number {0,number}.
      </messagebody>
   </message>

   <!--  -->
   <message
      messageid="VERSION"
      datelastchanged="967578656214"
      method="version(String version)"
      >
      <messagebody>
         Catalog version: {0}
      </messagebody>
   </message>

   <!--  -->
   <message
      messageid="I18N_PACKAGE"
      datelastchanged="967578691394"
      method="i18nPackage(String pkg)"
      >
      <messagebody>
         I18n Package: {0}
      </messagebody>
   </message>

   <!--  -->
   <message
      messageid="L10N_PACKAGE"
      datelastchanged="967578720156"
      method="l10nPackage(String pkg)"
      >
      <messagebody>
         L10n Package: {0}
      </messagebody>
   </message>

   <!--  -->
   <message
      messageid="SUBSYSTEM"


      datelastchanged="967578755587"
      method="subSystem(String sub)"
      >
      <messagebody>
         Catalog subsystem: {0}
      </messagebody>
   </message>
</message_catalog>
```

コード リスト 4-4 は、HelloWorld カタログを使用したアプリケーションの例を示しています。

コード リスト 4-4 HelloWorld カタログを使用したアプリケーションの例

```
package examples.i18n.simple;

import java.util.Locale;
import java.text.MessageFormat;

import weblogic.i18n.Localizer;
import weblogic.i18ntools.L10nLookup;

/**
 * @author Copyright (c) 2000 by BEA Systems, Inc. All Rights
Reserved.
 */

/**
 * この例では、シンプル メッセージ カタログを使用したアプリケーションを
 * インターナショナライズするさまざまな方法を示す
 * <p>
 * 使い方 : java examples.i18n.simple.HelloWorld [lang [country]]
 * <p>
 * lang: 「en」などの 2 文字の ISO 言語コード
 * country: 「US」などの 2 文字の ISO 国コード
 * <p>
 * この例でサポートされている言語を使用する場合、適切な OS
 * ローカライゼーション ソフトウェアと文字エンコーディングがあること
 * を前提とする
 * <p>
 * この例では、英語 ( デフォルト ) とフランス語のカタログが示されている
 * カタログ ソースは、以下のファイルにあり、カタログ編集ユーティリティ、
 * weblogic.i18ntools.gui.MessageEditor を使用して構築されたものである
 * <p>
 * <pre>
 * 英語 ( 基本言語 )          ../msgcat/Helloworld.xml
 * フランス語             ../msgcat/fr/FR/HelloWorld.xml
 * </pre>
 * <p>
 * このサンプルを構築するには、examples/i18n/simple ディレクトリから
bld.sh(UNIX)
 * または bld.cmd (NT) を実行する。このサンプルを実行する場合、
CLIENT_CLASSES
 * をセットアップし、これをクラスパスに配置する必要がある
 */


public final class HelloWorld {

  public static void main(String[] argv) {
```

```
    /*
     * ローカライズされたテキストを表示する最も簡単な方法は、
     * 生成された HelloWorld カタログのフォーマッタ クラスをインスタンス化する
ことである
     * このクラスには、カタログで定義された各メッセージをローカライズしたテキ
ストを
     * 返すコンビニエンス メソッドが入っている。クラス名は、カタログ名に
     * 「TextFormatter」を付加した名前になる
     *
     * 通常、現行ロケールでのフォーマットには、デフォルトの
     * コンストラクタを使用する。この例では、引数に基づいたロケールを使用して
     * TextFormatter を作成する
     */
    Locale lcl;
    if (argv.length == 0) { // デフォルトは JVM のデフォルト ロケール
      lcl = Locale.getDefault();
    }
    else {
      String lang = null;
      String country = null;
      // 言語コードを取得
      lang = argv[0];
      if (argv.length >= 2) { // 国コードを取得
        country = argv[1];
      }
      lcl = new Locale(lang,country);
    }
    /*
     * 適切なロケールによるフォーマッタを取得


     */
    HelloWorldTextFormatter fmt = new HelloWorldTextFormatter(lcl);
    fmt.setExtendedFormat(true);
    /*
     * 現行ロケールでテキストを出力
     */
    System.out.println(fmt.helloWorld());
    /*
     * また、手動でテキストにアクセスし、フォーマットすることもできる。その場
合、
     * そのカタログの Localizer クラスを取得する必要がある。Localizer クラ
スは
     * カタログの l10n_package 属性、カタログ名、および文字列
「TextLocalizer」
     * で構成される
     */
    Localizer l10n = L10nLookup.getLocalizer
      (lcl,"examples.i18n.simple.HelloWorldTextLocalizer");
    System.out.println(l10n.get("HELLO_AGAIN"));
    /*
     * メッセージに引数が入る場合、引数はメッセージに定義された
     * メソッドに渡されるだけである
     */
```

```
    System.out.println(fmt.nthHello(3));
    /*
     * マニュアル メソッドを使用する場合、MessageFormat クラスを使用して、引
数を手動で
     * テキストに適用する必要がある
     */
    String text = l10n.get("NTH_HELLO");
    Object[] args = {new Integer(4)};
    System.out.println(MessageFormat.format(text,args));
    /*
     * また、Localizer クラスは、カタログ情報にアクセスするメソッドも提供する
     */
    System.out.println(fmt.version(l10n.getVersion()));
    System.out.println(fmt.l10nPackage(l10n.getL10nPackage()));
    System.out.println(fmt.i18nPackage(l10n.getI18nPackage()));
    System.out.println(fmt.subSystem(l10n.getSubSystem()));
  }
}
```

コード リスト 4-5 は、HelloWorld カタログに対して生成された TextFormatter の例を示しています。

コード リスト 4-5 HelloWorld カタログに対して生成された TextFormatter クラスの例

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls600/msgcat.dtd">
<message_catalog

   l10n_package="examples.i18n.simple"

   subsystem="I18N"

   version="1.0"

   >

   <message
      messageid="HELLO_WORLD"
      datelastchanged="967575717875"
      method="helloWorld()"
      >
      <messagebody>
         Hello World!
      </messagebody>
   </message>
```

```
<!--  -->
<message
   messageid="HELLO_AGAIN"
   datelastchanged="967575717804"
   method="helloAgain()"
   >
   <messagebody>
      Hello again
   </messagebody>
</message>

<!--  -->
<message
   messageid="NTH_HELLO"
   datelastchanged="967575770971"
   method="nthHello(int count)"
   >
   <messagebody>
      This is hello number {0,number}.
   </messagebody>
</message>

<!--  -->
<message
   messageid="VERSION"
   datelastchanged="967578656214"
   method="version(String version)"
   >
   <messagebody>
      Catalog version: {0}
   </messagebody>
</message>

<!--  -->
<message
   messageid="I18N_PACKAGE"
   datelastchanged="967578691394"
   method="i18nPackage(String pkg)"
   >
   <messagebody>
      I18n Package: {0}
   </messagebody>
</message>

<!--  -->
<message
   messageid="L10N_PACKAGE"
   datelastchanged="967578720156"
   method="l10nPackage(String pkg)"
   >
   <messagebody>
      L10n Package: {0}
   </messagebody>
</message>

<!--  -->
```

```
   <message
      messageid="SUBSYSTEM"
      datelastchanged="967578755587"
      method="subSystem(String sub)"
      >
      <messagebody>
         Catalog subsystem: {0}
      </messagebody>
   </message>
</message_catalog>
```

索引

ろ