Oracle® Application Server 10*g*

Globalization Guide

10*g* (9.0.4)

Part No. B10380-01

September 2003

ORACLE®

Oracle Application Server 10*g* Globalization Guide, 10*g* (9.0.4)

Part No. B10380-01

Primary Author: Theresa M. Robertson

Contents

Send Us Your Comments ix

Preface xi

Audience xi
Documentation Accessibility xi
Organization xii
Related Documentation xiii
Conventions xiv

1 Overview of Globalization Support in Oracle Application Server

Globalizing Internet Applications 1-1
Globalization Concepts 1-1
Locale 1-2
Character Set 1-2
Unicode 1-2
Designing a Global Internet Application 1-2
Monolingual Internet Application Architecture 1-3
Multilingual Internet Application Architecture 1-4
Overview of Developing Global Internet Applications 1-6

Part I Development

2 Developing Locale Awareness

Developing Locale Awareness in Global Internet Applications 2-1
Locale Awareness in J2EE and Internet Applications 2-4
Locale Awareness in Java Applications 2-5
Locale Awareness in Perl and C/C++ Applications 2-6
Locale Awareness in SQL and PL/SQL Applications 2-7
Locale Awareness in Oracle Application Server Component Applications 2-9
Locale Awareness in Oracle Application Server Wireless Services 2-9
Locale Awareness in Oracle Application Server Forms Services 2-9
Locale Awareness in Oracle Application Server Reports Services 2-12
Locale Awareness in Oracle Application Server Discoverer 2-14

3 Implementing HTML Features

Implementing HTML Features for Global Applications 3-1
Encoding HTML Pages 3-1
Specifying the Page Encoding for HTML Pages 3-3
Specifying the Encoding in the HTTP Header 3-4
Specifying the Encoding in the HTML Page Header 3-4
Specifying the Page Encoding in Java Servlets and Java Server Pages 3-4
Specifying the Page Encoding in PL/SQL Server Pages 3-5
Specifying the Page Encoding in Perl 3-7
Specifying the Page Encoding in Oracle Application Server Mobile Services Applications 3-8
Specifying the Page Encoding in Oracle Application Server Web Cache Enabled Applications 3-9
Specifying the Page Encoding in Oracle Application Server Reports Services Applications 3-10
Specifying the Page Encoding in JSP Reports for the Web 3-10
Specifying the Page Encoding in HTML for Oracle Application Server Reports Services 3-10
Specifying the Page Encoding in XML for Oracle Application Server Reports Services 3-11

Handling HTML Form Input ... 3-11
Handling HTML Form Input in Java ... 3-12
Handling HTML Form Input in PL/SQL ... 3-13
Handling HTML Form Input in Perl ... 3-14
Handling Form Input in Oracle Application Server Mobile Services Applications ... 3-15
Decoding HTTP Headers ... 3-16
Decoding HTTP Headers from Oracle Application Server Single Sign-On ... 3-16
Decoding String-type Mobile Context Information Headers in Oracle Application Server Wireless Services ... 3-17
Encoding URLs ... 3-17
Encoding URLs in Java ... 3-18
Encoding URLs in PL/SQL ... 3-19
Encoding URLs in Perl ... 3-19
Formatting HTML Pages to Accommodate Text in Different Languages ... 3-20
Organizing the Content of HTML Pages for Translation ... 3-21
Translation Guidelines for HTML Page Content ... 3-21
Organizing Static Files for Translation ... 3-22
Organizing Translatable Static Strings for Java Servlets and Java Server Pages ... 3-23
Organizing Translatable Static Strings in C/C++ and Perl ... 3-26
Organizing Translatable Static Strings in Message Tables ... 3-27
Organizing Translatable Dynamic Content in Application Schema ... 3-28

4 Using a Centralized Database

Using a Centralized Database and Accessing the Database Server ... 4-1
Using JDBC to Access the Database ... 4-2
Using PL/SQL to Access the Database ... 4-3
Using Perl to Access the Database ... 4-4
Using C/C++ to Access the Database ... 4-5
Using the OCI API to Access the Database ... 4-6
Using the Unicode API Provided with OCI to Access the Database ... 4-7
Using Unicode Bind and Define in Pro*C/C++ to Access the Database ... 4-8

Part II Deployment

5 Configuring Oracle Application Server for Global Deployment

Installing Oracle Application Server for Global Deployment 5-1
Configuring Oracle HTTP Server and OC4J for Global Deployment 5-3
About Manually Editing HTTP Server and OC4J Configuration Files 5-3
Configuring the NLS_LANG Parameter 5-4
Preconfigured NLS_LANG Values 5-6
Configuring Transfer Mode for mod_plsql Runtime 5-9
Configuring the Runtime Default Locale 5-10
mod_jserv Runtime for Java 5-10
OC4J Java Runtime 5-11
mod_plsql Runtime for PL/SQL and PL/SQL Server Pages 5-11
mod_perl Runtime for Perl Scripts 5-11
C/C++ Runtime 5-11
Configuring Oracle Application Server Portal for Global Deployment 5-12
Configuring Oracle Application Server Wireless for Global Deployment 5-13
Configuring Encoding for Outgoing Email Messages 5-13
Configuring Oracle Application Server Single Sign-On for Global Deployment 5-13
Configuring Oracle Application Server Forms Services for Global Deployment 5-14
Configuring Oracle Application Server Reports Services for Global Deployment 5-15
Configuring Oracle Application Server Discoverer for Global Deployment 5-16
Configuring Oracle Business Components for Java for Global Deployment 5-16
Configuring a Centralized Unicode-enabled Database to Support Global Deployment 5-17

6 A Multilingual Demo for Oracle Application Server

Description of the World-of-Books Demo 6-1
Architecture and Design of the World-of-Books Demo 6-2
World-of-Books Architecture 6-2
World-of-Books Design 6-4
World-of-Books Schema Design 6-5
Installing the World-of-Books Demo 6-7

Building, Deploying, and Running the World-of-Books Demo 6-7
How to Build the World-of-Books Demo 6-9
How to Deploy the World-of-Books Demo 6-10
How to Run the World-of-Books Demo 6-11
Locale Awareness of the World-of-Books Demo 6-12
How World-of-Books Determines the User's Locale 6-13
How World-of-Books Uses Locale Information in Localizer Methods 6-14
How World-of-Books Sorts Query Results 6-15
How World-of-Books Searches the Contents of Books 6-16
Encoding HTML Pages for the World-of-Books Demo 6-17
Handling HTML Form Input for the World-of-Books Demo 6-17
Encoding URLs in the World-of-Books Demo 6-18
Formatting HTML Pages in the World-of-Books Demo 6-19
Accessing the Database in the World-of-Books Demo 6-20
Organizing the Content of HTML Pages in the World-of-Books Demo 6-20
Static Files for World-of-Books Online Help 6-21
Using Resource Bundles for the Content of World-of-Books HTML Pages 6-21

A Oracle Application Server Translated Languages

Glossary

Index

Send Us Your Comments

Oracle Application Server 10*g* Globalization Guide, 10*g* (9.0.4)

Part No. B10380-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: appserverdocs_us@oracle.com
- FAX: 650-506-7375 Attn: Oracle Application Server Documentation Manager
- Postal service:
  Oracle Corporation
  Oracle Application Server Documentation
  500 Oracle Parkway, M/S 1op6
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, please contact your local Oracle Support Services.

Preface

Oracle Application Server 10g Globalization Guide describes how to design, develop, and deploy Internet applications for a global audience.

This preface contains the following topics:

- Audience
- Documentation Accessibility
- Organization
- Related Documentation
- Conventions

Audience

Oracle Application Server 10g Globalization Guide is intended for Internet application developers and Webmasters who design, develop, and deploy Internet applications for a global audience.

To use this document, you need to have some programming experience and be familiar with Oracle databases.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to

evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Organization

This document contains:

Chapter 1, "Overview of Globalization Support in Oracle Application Server"
This chapter defines concepts that are essential to understanding the rest of the book. It also describes models for monolingual Internet application design and multilingual Internet application design.

Part I, "Development"

Chapter 2, "Developing Locale Awareness"
This chapter describes how to make Internet applications locale-aware and how to present locale-appropriate data to users.

Chapter 3, "Implementing HTML Features"
This chapter describes how to encode HTML pages, handle HTML form input, and encode URLs so that clients in different locales can exchange information with the application server.

Chapter 4, "Using a Centralized Database"

This chapter describes how the application server accesses the database with minimal character set conversion and data loss.

Part II, "Deployment"

Chapter 5, "Configuring Oracle Application Server for Global Deployment"

This chapter describes how to configure Oracle Application Server (Oracle Application Server) for global application deployment.

Chapter 6, "A Multilingual Demo for Oracle Application Server"

This chapter describes World-of-Books, the multilingual demo that is provided with Oracle Application Server.

Appendix A, "Oracle Application Server Translated Languages"

This appendix contains a list of languages that Oracle Application Server supports.

Glossary

The glossary defines terms that are related to globalization support for Oracle Application Server.

Related Documentation

For more information, see these Oracle resources:

- The Oracle Application Server documentation set
- *Oracle9i Globalization Support Guide* in the Oracle Database Documentation Library

Printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://otn.oracle.com/membership`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

http://otn.oracle.com/documentation/

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width font)`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.

Convention	Meaning	Example
`lowercase monospace (fixed-width font)`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the data files and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`. Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase monospace (fixed-width font) italic`	Lowercase monospace italic font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
`[ ]`	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*`digits`*` [ , `*`precision`*` ])`
`{ }`	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`
`|`	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`

Convention	Meaning	Example
. . .	Horizontal ellipsis points indicate either: ■ That we have omitted parts of the code that are not directly related to the example ■ That you can repeat a portion of the code	`CREATE TABLE ... AS subquery;` `SELECT col1, col2, ... , coln FROM employees;`
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);` `acct CONSTANT NUMBER(4) := 3;`
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/system_password` `DB_NAME = database_name`

UPPERCASE	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

lowercase	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

1

Overview of Globalization Support in Oracle Application Server

This chapter contains the following topics:

- Globalizing Internet Applications
- Globalization Concepts
- Designing a Global Internet Application
- Overview of Developing Global Internet Applications

Globalizing Internet Applications

It is increasingly important for businesses to make their Internet applications available to users around the world with appropriate locale characteristics, such as language and currency formats. The Oracle Application Server (Oracle Application Server) is fully internationalized to provide a global platform for developing and deploying Internet applications.

Building an Internet application or Web site for Oracle Application Server requires good globalization practices in development and deployment. This book describes recommended globalization practices.

Chapter 6 contains information about a multilingual demo that is included with Oracle Application Server.

Globalization Concepts

You need to be familiar with the following concepts to understand the rest of this book:

- Locale
- Character Set
- Unicode

Locale

Locale refers to a language, a character set, and the region (territory) in which the language is spoken. Information about the region includes formats for dates and currency. For example, the primary languages of the United States and Great Britain are both forms of English, but the two territories have different currencies and different conventions for date formats. Therefore, the United States and Great Britain are different locales.

Character Set

A **character set** defines the binary values that are associated with the characters that make up a language. For example, the ISO-8859-1 character set can be used to encode most Western European languages.

Unicode

Unicode is a universal character set that defines binary values for characters in almost all languages. Unicode characters can be encoded as follows:

- In 1 to 4 bytes in the UTF-8 character set
- In 2 or 4 bytes in the UTF-16 character set
- In 4 bytes in the UTF-32 character set

Designing a Global Internet Application

There are several approaches to designing global Internet applications. This book discusses two approaches: monolingual and multilingual.

You can design a monolingual Internet application so that it supports several instances. Each instance supports a different locale. Users with different locale preferences must invoke the instance that serves their locale.

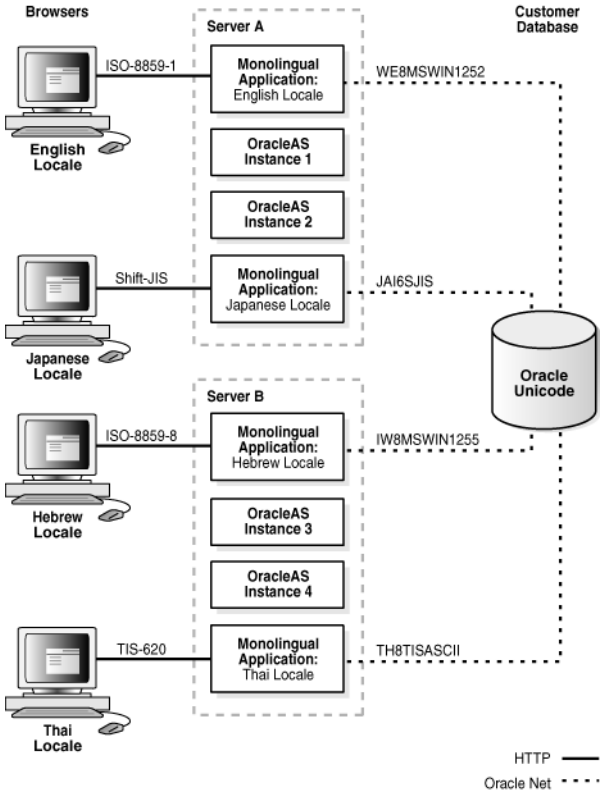
You can design a multilingual Internet application to support several locales with one instance. All users, regardless of locale, can invoke the same instance.

Both designs include one centralized database that uses a Unicode character set.

Monolingual Internet Application Architecture

Figure 1–1 shows the design of a monolingual Internet application.

Figure 1–1 Monolingual Internet Application Architecture



The clients (in English, Japanese, Hebrew, and Thai locales) communicate with separate instances of Oracle Application Server through HTTP connections. One instance of the application runs in the same locale as one of the Oracle Application Server instances. For example, the English application runs in the same locale as Oracle Application Server Instance 1. The English and Japanese applications and their Oracle Application Server instances are running on Server A, and the Hebrew and Thai applications and their instances are running on Server B. Each Oracle Application Server instance communicates with the Unicode database. The instances communicate with the database through Oracle Net.

The client character set for the English locale, for example, is ISO-8859-1. The Oracle Application Server instance that is associated with the English locale, Instance 1, uses the Oracle character set WE8MSWIN1252 to communicate with the database. The database character set is a Unicode character set.

See Also: Chapter 4, "Using a Centralized Database"

Table 1–1 shows the advantages and disadvantages of deploying monolingual Internet applications. As the number of locales increases, the disadvantages outweigh the advantages of the monolingual design. This type of application design is suitable for customers who support only one or two locales.

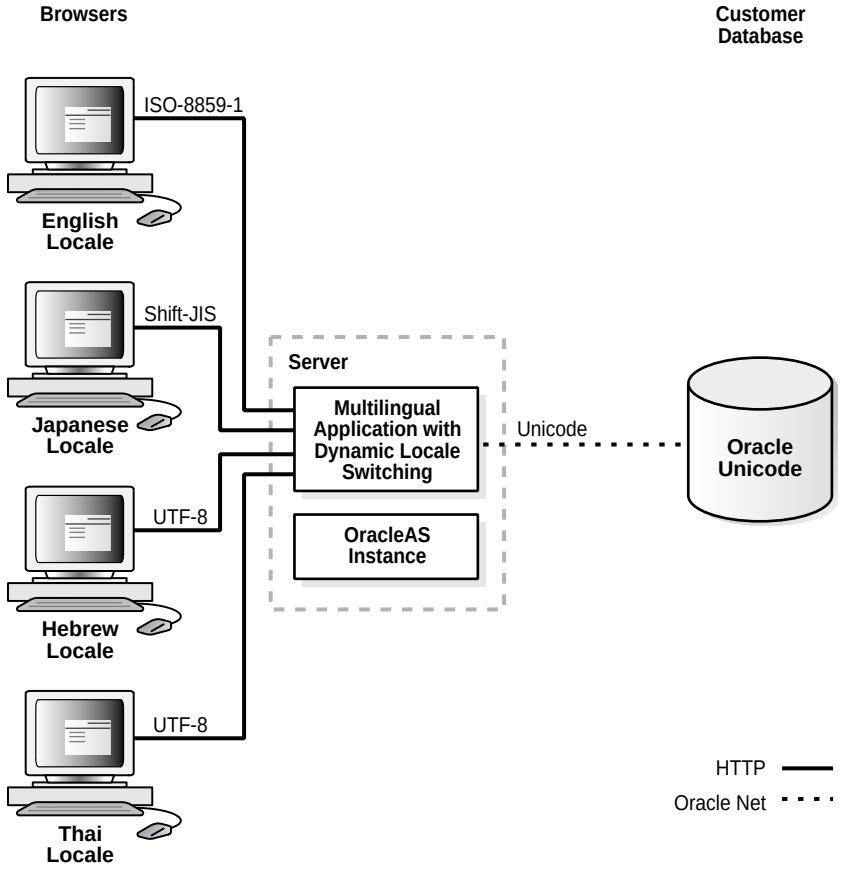
Table 1–1 Advantages and Disadvantages of Monolingual Internet Application Design

Advantages	Disadvantages
You can separate the support of different locales into different servers. This allows locales to be supported in different time zones. Work load can be distributed accordingly.	There are more Oracle Application Server servers to administer.
Writing the code is simpler than for a multilingual Internet application.	The Internet application requires more testing resources to certify it on each Oracle Application Server instance. You must configure Oracle Application Server for each instance of the application. You must maintain a server instance for each locale regardless of the amount of work that is demanded of it. Load-balancing is possible only among a group of Oracle Application Server instances that support the same locale. Supporting multilingual content is difficult.

Multilingual Internet Application Architecture

Figure 1–2 shows the design of a multilingual Internet application.

Figure 1–2 Multilingual Internet Application Architecture



The clients (in English, Japanese, Hebrew, and Thai locales) communicate with one Oracle Application Server instance through HTTP connections. Each client can use a different character set because each application running on Oracle Application Server is configured to support several locales simultaneously, regardless of the locale of the Oracle Application Server instance. The Oracle Application Server instance and the database communicate through Oracle Net. Both the application running on the Oracle Application Server instance and the database use Unicode character sets.

See Also: Chapter 4, "Using a Centralized Database"

In order to support several locales in a single application instance, an application should:

- Process character data in Unicode so that it can support data in any language
- Dynamically detect the user's locale and adapt to the locale by constructing HTML pages in the correct language and cultural conventions
- Dynamically determine the character set to use for HTML pages and convert content from Unicode to the HTML page encoding and vice versa

Table 1–2 shows the advantages and disadvantages of deploying multilingual Internet applications.

Table 1–2 Advantages and Disadvantages of Multilingual Internet Application Design

Advantages	Disadvantages
You can use one Oracle Application Server configuration, which reduces maintenance costs.	Multilingual applications are more complex to code than monolingual applications. They must be able to detect locales dynamically and use Unicode. This is costly if you only need to support one or two languages.
Performance tuning and capacity planning do not depend on the number of locales.	
Supporting additional languages is relatively easy. You do not need to add more machines for the new locales.	
You can test the application for several locales in a single testing environment.	
The application can support multilingual content.	

Overview of Developing Global Internet Applications

Building an Internet application for Oracle Application Server that supports different locales requires good development practices. The application itself must be aware of the user's locale and be able to present locale-appropriate content to the user. Clients must be able to communicate with the application server regardless of the client's locale, with minimal character set conversion. The application server must be able to access the database server with data in many languages, again with minimal character set conversion. Character set conversion decreases performance and increases the chance of data loss because some characters may not be available in the target character set.

> **See Also:** *Oracle9i Globalization Support Guide* in the Oracle Database Documentation Library for more information about character set conversion

Oracle Application Server supports the following programming languages and corresponding Web development environments for developing global Internet and J2EE applications:

- Java callable from the `mod_oc4j` module

  Oracle Application Server Containers for J2EE (OC4J) is the Java2 Enterprise Edition (J2EE) container that Oracle Application Server provides. The `mod_oc4j` module routes HTTP requests to Java Servlets and JSPs running on the OC4J Servlet runtime.

- Java callable from the `mod_jserv` module

  The `mod_jserv` module routes HTTP requests to the corresponding Java Servlets and Java Server Pages (JSPs) running on the Jserv Servlet runtime.

- Perl callable from the `mod_perl` module

  The standard `mod_perl` module provides fast execution of Perl scripts. The `mod_perl` module routes HTTP requests to the Perl scripts running on the Perl interpreter from within the `mod_perl` module.

- PL/SQL callable from the `mod_plsql` module

  The `mod_plsql` module is a gateway that routes HTTP requests from the Web server to PL/SQL procedures or PL/SQL Server Pages (PSP) running in a database.

- C/C++ as CGI or shareable libraries

  You can write C and C++ programs as CGI applications callable from the `mod_fastcgi` module or as libraries callable from any of the above Web development environments.

> **See Also:** *Oracle HTTP Server Administrator's Guide* for more information about the modules

---

Note: In this book, **encoding** and **page encoding** refer to the character set used in a particular programming environment.

Oracle Application Server also supports the development of global applications using the following Oracle Application Server components:

- Oracle Application Server Forms Services
- Oracle Application Server Reports Services
- Oracle Application Server Discoverer
- Oracle Application Server Web Cache
- Oracle Application Server Wireless

This guide discusses global application development in terms of each of these languages and development environments. It addresses the basic tasks associated with developing and deploying global Internet applications, including developing locale awareness, implementing HTML features, accessing a centralized database, and configuring Oracle Application Server.

Part I

Development

Part I contains the following chapters:

- Chapter 2, "Developing Locale Awareness"
- Chapter 3, "Implementing HTML Features"
- Chapter 4, "Using a Centralized Database"

2

Developing Locale Awareness

This chapter contains the following topics:

- Developing Locale Awareness in Global Internet Applications
- Locale Awareness in J2EE and Internet Applications
- Locale Awareness in Oracle Application Server Component Applications

Developing Locale Awareness in Global Internet Applications

Global Internet applications need to be aware of the user's locale.

Locale-sensitive functions, such as date formatting, are built into programming environments such as C/C++, Java, and PL/SQL. Applications can use locale-sensitive functions to format the HTML pages according to the cultural conventions of the user's locale.

Different programming environments represent locales in different ways. For example, the French (Canada) locale is represented as follows:

Environment	Representation	Locale	Explanation
Various	ISO standard	`fr-CA`	`fr` is the language code defined in the ISO 639 standard. `CA` is the country code defined in the ISO 3166 standard.
Java	Java locale object	`fr_CA`	Java uses the ISO language and country code.

Environment	Representation	Locale	Explanation
C/C++	POSIX locale name	`fr_CA` on Sun Solaris	POSIX locale names may include a character set that overrides the default character set. For example, the de.ISO8859-15 locale is used to support the Euro symbol.
PL/SQL and SQL	`NLS_LANGUAGE` and `NLS_TERRITORY` parameters	`NLS_LANGUAGE= "CANADIAN FRENCH"` `NLS_TERRITORY= "CANADA"`	**See Also:** Chapter , "Configuring the NLS_LANG Parameter"

Table 2–1 shows how different programming environments represent some commonly used locales.

Table 2–1 Locale Representations in Different Programming Environments

Locale	ISO	Java	POSIX Solaris	NLS_LANGUAGE, NLS_TERRITORY
Arabic (U.A.E.)	ar	ar	ar	ARABIC, UNITED ARAB EMIRATES
Germany (German)	de-DE	de_DE	de	GERMANY, GERMAN
English (U.S.A)	en	en_US	en_US	AMERICAN, AMERICA
English (United Kingdom)	en-GB	en_GB	en_UK	ENGLISH, UNITED KINGDOM
Greek	el	el	el	GREEK, GREECE
Spanish (Spain)	es-ES	es_ES	es	SPANISH, SPAIN
French (France)	fr	fr_FR	fr	FRENCH, FRANCE
French (Canada)	fr-CA	fr_CA	fr_CA	CANADIAN FRENCH, CANADA
Hebrew	he	he	he	HEBREW, ISRAEL
Italian (Italy)	it	it	it	ITALIAN, ITALY
Japanese	ja-JP	ja_JP	ja_JP	JAPANESE, JAPAN

Table 2–1 Locale Representations in Different Programming Environments (Cont.)

Locale	ISO	Java	POSIX Solaris	NLS_LANGUAGE, NLS_TERRITORY
Korean	ko-KR	ko_KR	ko_KR	KOREAN, KOREA
Portuguese (Portugal)	pt	pt	pt	PORTUGUESE, PORTUGAL
Portuguese (Brazil)	pt-BR	pt_BR	pt_BR	BRAZILIAN PORTUGUESE, BRAZIL
Turkish	tr	tr	tr	TURKISH, TURKEY
Thai	th	th	th	THAI, THAILAND
Chinese (Taiwan)	zh-TW	zh_TW	zh_TW	TRADITIONAL CHINESE, TAIWAN
Chinese (P.R.C)	zh-CN	zh_CN	zh_CN	SIMPLIFIED CHINESE, CHINA

If you write applications for more than one programming environment, then locales must be synchronized between environments. For example, Java applications that call PL/SQL procedures should map the Java locales to the corresponding `NLS_LANGUAGE` and `NLS_TERRITORY` values and change the parameter values to match the user's locale before calling the PL/SQL procedures.

There are two thing that affect an application's overall locale awareness: the development environment in which you create the application, and the target architecture for which the application is built. This chapter addresses these topics with respect to both monolingual and multilingual application architectures.

Determining a User's Locale in Monolingual Internet Applications

A monolingual application, by definition, serves users with the same locale. A user's locale is fixed in a monolingual application and is the same as the default runtime locale of the programming environment.

In almost all programming environments, almost all locale-sensitive functions implicitly use the default runtime locale to perform their tasks. Monolingual applications can rely on this behavior when calling these functions.

Determining a User's Locale in Multilingual Internet Applications

In a multilingual application, the user's locale may vary. Multilingual applications should do the following:

- Dynamically detect the user's locale

- Construct HTML content in the language of the locale
- Use the cultural conventions implied by the locale

Multilingual applications can determine a user's locale dynamically in the following ways:

- Based on the user profile information from an LDAP directory server such as Oracle Internet Directory (OID)

  The application can store the user profile in the OID server provided by Oracle Application Server. The LDAP schema for the user profile should include a preferred locale attribute. This method does not work if a user has not logged on before.

- Based on the default ISO locale of the user's browser

  Every HTTP request sends the default ISO locale of the browser with the Accept-Language HTTP header. If the Accept-Language header is NULL, then the locale should default to English. The drawback of this approach is that the Accept-Language header may not be a reliable source of information about the user's locale.

- Based on user input

  Users can select a locale from a list or a group of icons such as flags.

You can use these methods of determining the user's locale together or separately. After the application determines the locale, the locale should be:

- Mapped to the locale representations that correspond to the programming environments on which the application runs
- Used in locale-sensitive functions

> **See Also:** Table 2–1 for common locale representations in different programming environments

Locale Awareness in J2EE and Internet Applications

This section discusses locale awareness in terms of the particular programming language and development environment in which an application is written.

Locale Awareness in Java Applications

A Java locale object represents the corresponding user's locale in Java. The Java encoding used for the locale is required to properly convert Java strings to byte data and vice versa.

Consider the Java encoding for the locale when you make the Java code aware of a user's locale. There are two ways to make a Java method sensitive to the Java locale and the Java encoding:

- Using the default Java locale and default Java encoding for the method
- Explicitly specifying the Java locale and Java encoding for the method

Locale Awareness in Monolingual Java Applications

Monolingual applications should run implicitly with the default Java locale and default Java encoding so that the applications can be configured easily for a different locale. For example, to create a date formatter using the default Java locale, use the following method call:

```
DateFormat df = DateFormat.getDateTimeInstance(DateFormat.FULL, DateFormat.FULL);
dateString = df.format(date); /* Format a date */
```

Locale Awareness in Multilingual Java Applications

You should develop multilingual applications such that they are independent of fixed default locales or encodings. Explicitly specify the Java locale and Java encoding that correspond to the current user's locale. For example, specify the Java locale object that corresponds to the user's locale, identified by `user_locale`, in the `getDateTimeInstance()` method:

```
DateFormat df = DateFormat.getDateTimeInstance(DateFormat.FULL, DateFormat.FULL, user_
locale);
dateString = df.format(date); /* Format a date */
```

Note that the only difference between the example code for the monolingual application and the multilingual application is the inclusion of `user_locale`.

Similarly, do not use encoding-sensitive methods that assume the default Java encoding. For example, you should not use the `String.getBytes()` method in a multilingual application because it is encoding-sensitive. Instead, use the method that accepts encoding as an argument, which is `String.getBytes(String encoding)`. Be sure to specify the encoding used for the user's locale.

Do not use the `Locale.setDefault()` method to change the default locale for these reasons:

- It changes the Java default locale for all threads and makes your applications unsafe to threads
- It does not affect the Java default encoding

Locale Awareness in Perl and C/C++ Applications

Perl and C/C++ use the POSIX locale model for internationalized applications.

Locale Awareness in Monolingual Perl and C/C++ Applications

Monolingual applications should be sensitive to the default POSIX locale, which is configured by changing the value of the `LC_ALL` environment variable or changing the operating system locale from the Control Panel in Windows.

See Also: Table 2–1 for a list of commonly used POSIX locales

To run on the default POSIX locale, the applications should call the `setlocale()` function to set the default locale to the one defined by `LC_ALL` and use the POSIX locale-sensitive functions such as `strftime()` thereafter. Note that the `setlocale()` function affects the current process and all the threads associated with it, so any multithread application should assume the same POSIX locale in each thread. The following example gets the current time in the format specific to the default locale in Perl:

```
use locale;
use POSIX qw (locale_h);
...
$old_locale = setlocale( LC_ALL, "" );
$dateString = POSIX::strftime( "%c", localtime());
...
```

Locale Awareness in Multilingual Perl and C/C++ Applications

Multilingual applications should be sensitive to dynamically determined locales. Call the `setlocale()` function to initialize the locale before calling locale-sensitive functions. For example, the following C code gets the local time in the format of the user locale identified by `user_locale`:

```
#include <locale.h>
#include <time.h>
    ...
    const char *user_locale = "fr";
    time_t ltime;
    struct tm *thetime;
```

```
unsigned char dateString[100];
...
setlocale(LC_ALL, user_locale);
time (&ltime);
thetime = gmtime(&ltime);
strftime((char *)dateString, 100, "%c", (const struct tm *)thetime));
...
```

You must map user locales to POSIX locale names for applications to initialize the correct locale dynamically in C/C++ and Perl. The POSIX locales depend on the operating system.

Locale Awareness in SQL and PL/SQL Applications

PL/SQL procedures run in the context of a database session whose locale is initialized by the NLS_LANG parameter in the **database access descriptor (DAD)**. The NLS_LANG parameter specifies top-level NLS parameters, NLS_LANGUAGE and NLS_TERRITORY, for the database session. Other NLS parameters, such as NLS_SORT and NLS_DATE_LANGUAGE, inherit their values from these top-level parameters. These NLS parameters define the locale of a database session.

See Also:

- Chapter, "Configuring the NLS_LANG Parameter"
- *Oracle9i Database Reference* in the Oracle Database Documentation Library
- *Oracle9i Globalization Support Guide* in the Oracle Database Documentation Library

for more information about NLS parameters

There are two ways to make SQL and PL/SQL functions locale sensitive:

- Basing the locale on the NLS parameters of the current database session
- Explicitly specifying the NLS parameters

Locale Awareness in Monolingual SQL and PL/SQL Applications

Generally speaking, the initial values of the NLS parameters inherited from NLS_LANG are sufficient for monolingual PL/SQL procedures. For example, the following PL/SQL code calls the TO_CHAR() function to get the formatted date, which uses the current values of the NLS_DATE_FORMAT and NLS_DATE_LANGUAGE parameters:

```
mydate date;
```

```
dateString varchar2(100);
...
select sysdate into mydate from dual;
dateString = TO_CHAR(mydate);
```

If the initial values of the NLS parameters are not appropriate, then use an ALTER SESSION statement to overwrite them for the current database session. You can use the ALTER SESSION statement with the DBMS_SQL package. For example:

```
cur integer;
status integer;
...
cur := dbms_sql.open_cursor;
dbms_sql.parse(cur, 'alter session set nls_date_format = "Day Month, YYYY"',
       dbms_sql.native);
status := dbms_sql.execute(cur);
```

Locale Awareness in Multilingual SQL and PL/SQL Applications

Multilingual applications should use ALTER SESSION statements to change the locale of the database session to the user's locale before calling any locale-sensitive SQL or PL/SQL functions. You can use the ALTER SESSION statement with the DBMS_SQL package. For example:

```
cur integer;
status integer;
...
cur := dbms_sql.open_cursor;
dbms_sql.parse(cur, 'alter session set nls_language = "NLS_LANGUAGE_of_user_
       locale"', dbms_sql.native);
dbms_sql.parse(cur, 'alter session set nls_territory = "NLS_TERRITORY_of_
       user_locale"', dbms_sql.native);
status := dbms_sql.execute(cur);
```

Alternatively, applications can specify the NLS parameters in every SQL function that accepts an NLS parameter as an argument. For example, the following PL/SQL code gets a date string based on the language of the user's locale:

```
mydate date;
dateString varchar2(100);
...
select sysdate into mydate from dual;
dateString TO_CHAR(mydate, 'DD-MON-YYYY HH24:MI:SSxFF',
                    'NLS_DATE_LANGUAGE=language' );
...
```

language specifies the Oracle language name for the user's locale.

Locale Awareness in Oracle Application Server Component Applications

This section discusses locale awareness in terms of application development for particular Oracle Application Server components.

Locale Awareness in Oracle Application Server Wireless Services

Oracle Application Server Wireless sends all of the Mobile Context information as HTTP headers when invoking a request. The user locale is sent using the `X-Oracle-User.Locale` header. The locale value contains the ISO language, and an optional ISO country code, separated by a hyphen. For example, "en-US", "zh-CN", and "ja" are all valid locale values for this header. Mobile service applications should use the user locale specified in this header to determine the language and cultural conventions used in the user interface.

For example, JSP applications may retrieve the user locale as follows:

```
<%
   String userLocale = request.getHeader("X-Oracle-User.Locale");
%>
```

Locale Awareness in Oracle Application Server Forms Services

The Oracle Application Server Forms Services architecture includes:

- A Java Client (browser)
- Oracle Application Server Forms Services (middle tier)
- The Oracle9*i* customer database (back end)

The Java Client is dynamically downloaded from Oracle Application Server when a user runs a Forms Services session. The Java Client provides the user interface for the Forms Services Runtime Engine. It also handles user interaction and visual feedback for actions such as navigating between items or checking a checkbox.

Oracle Application Server Forms Services consists of the Forms Services Runtime Engine and the Forms Listener Servlet. The Forms Services Runtime Engine is the process that maintains a connection to the database on behalf of the Java Client. The Forms Listener Servlet acts as a broker, taking connection requests from the Java Client processes and initiating a Forms Services runtime process on their behalf.

The `NLS_LANG` parameter for Forms Services initializes the locale of Oracle Application Server Forms Services. The `NLS_LANGUAGE` parameter derives its value

from NLS_LANG and determines the language of Forms messages. The NLS_TERRITORY parameter also derives its value from NLS_LANG and determines conventions such as date and currency formats.

By default, the NLS_LANG parameter for Oracle Application Server Forms Services initializes the Java Client locale. The locale of the Java Client determines such things as button labels on default messages and parts of strings in menus.

> **See Also:** *Oracle Application Server Forms Services Deployment Guide*

Locale Awareness in Monolingual Oracle Application Server Forms Services Applications

A user's locale is fixed in a monolingual Oracle Application Server Forms Services application and is usually the same as the default Forms Services locale. When you develop a monolingual Forms Services application, you must develop it to conform to the intended user's locale. The database character set should be a superset of the Forms Services character set.

For example, a monolingual Forms Services application for a Japanese locale should include Japanese text, Japanese button labels, and Japanese menus. The application should also connect to a database whose character set is JA16SJIS, JA16EUC, or UTF8.

The NLS_LANG parameter in the default.env file controls the Forms Services locale. Additionally, in order to pass non-Latin-1 parameters to Forms Services, you can set the defaultcharset parameter in formsweb.cfg.

> **See Also:** *Oracle Application Server Forms Services Deployment Guide*

Locale Awareness in Multilingual Oracle Application Server Forms Services Applications

In a multilingual environment, the application can dynamically determine the locale of Oracle Application Server Forms Services in two ways:

- Based on the user's profile
- Based on the user's input

When you develop a Forms Services application you must choose one of these methods.

You can configure multilingual Forms Services applications by using multiple environment configuration files (`EnvFile`). For example, you can create a form called `form.fmx` and translate it into Japanese and into Arabic using Oracle9*i* Translator. Then save them as `d:\form\ja\form.fmx` (Japanese) and `d:\form\ar\form.fmx` (Arabic). Finally, create two environment configurations files, `ja.env` and `ar.env`, and specify the following in the appropriate environment file:

Form	Environment File	NLS_LANG	FORMS90_PATH
d:\form\ja\form.fmx	ja.env	JAPANESE_JAPAN.JA16SJIS	d:\form\ja
d:\form\ar\form.fmx	ar.env	ARABIC_EGYPT.ARMSWIN1256	d:\form\ar

Also, you can configure Forms Services to read the preferred language settings of the browser. For example, if you have a human resources application translated into 24 languages, then add an application entry in the `formsweb.cfg` file like the following:

```
[HR]
default.env
[HR.DE]
DE.env
[HR.FR]
FR.env
.
.
.
```

When the Forms Servlet detects a language preference in the browser, it checks the `formsweb.cfg` file to see if there is a translated version of the application.

For example, suppose the request is `http://`*`myserver.mydomain`*`/servlet/f90servlet?config=HR` and the preferred languages are set to German (`DE`), Italian (`IT`), and French (`FR`), in this order, then this is the order of priority. The Forms Servlet tries to read from the application definitions in the following order:

```
HR.DE
HR.IT
HR.FR
HR
```

If the Forms Servlet cannot find any of those configurations, then it uses the HR configuration (default.env).

This means that you can configure Forms to support multiple languages with one URL. Each application definition can have its own environment file that contains the NLS language parameter definition. You can also specify separate working directory information and path information for each application.

> **See Also:** Chapter 5, "Configuring Oracle Application Server for Global Deployment"

Additionally, Forms can display more than one language in a form if the Java client machine has the Albany WT J font installed. You can obtain this font from the utilities CD in your CD pack or from http://metalink.oracle.com.

The Albany WT J font should be copied to %WINDOWS%\Fonts if the client is using JInitiator 1.3.1, or %JAVA_HOME%\lib\fonts if the client is using Java Plug-in 1.4.1.

Locale Awareness in Oracle Application Server Reports Services

The Oracle Application Server Reports Services architecture includes:

- A client tier (browser)
- A Reports Server (middle tier)
- An Oracle9*i* customer database (back end)

Oracle Application Server Reports Services can run multiple reports simultaneously upon users' requests. The Reports Server enters requests for reports into a job queue and dispatches them to a dynamic, configurable number of pre-spawned runtime engines. The runtime engine connects to the database, retrieves data, and formats output for the client.

The NLS_LANG setting for the Reports Server initializes the locale of the runtime engine. The NLS_LANGUAGE parameter derives its value from the NLS_LANG parameter and determines the language of the Reports Server messages. The NLS_TERRITORY parameter derives its value from the NLS_LANG parameter and determines the date and currency formats. For example, if NLS_LANG is set to JAPANESE_JAPAN.JA16SJIS, then Reports Server messages are in Japanese and reports use the Japanese date format and currency symbol.

The dynamic environment switching feature enables one instance of the Reports Server to serve reports with any arbitrary environment setting, including NLS_

LANG. Using the environment element in the Reports Server configuration file, you can create a language environment that can be referenced in two ways:

- On a per-runtime engine basis through the engine element of the Reports Server configuration file
- On a per-job basis using the ENVID command line argument

> **See Also:** *Oracle Application Server Reports Services Publishing Reports to the Web* for more information about dynamic environment switching

Report output is generated in the Reports Services character set. The client needs to be aware of the character set in which Reports Services generated the HTML or XML.

> **See Also:** "Specifying the Page Encoding in Oracle Application Server Reports Services Applications"

Locale Awareness in Monolingual Oracle Application Server Reports Services Applications

A user's locale is fixed in a monolingual Oracle Application Server Reports Services application and is usually the same as the locale of the Reports Server. The database character set should be a superset of the Reports Server character set.

Locale Awareness in a Multilingual Oracle Application Server Reports Services Application

In a multilingual report, the application can dynamically determine the locale of the Reports Server in two ways:

- Based on the user's profile
- Based on the user's input

When you develop a report you must choose one of these methods.

You can use the dynamic environment switching feature to support multiple languages.

See Also:

- Chapter , "Specifying the Page Encoding in HTML for Oracle Application Server Reports Services" for more information about the encoding of HTML, XML, and JSP report output
- Chapter , "Configuring Oracle Application Server Reports Services for Global Deployment" for more information about specifying NLS_LANG parameters from the command line
- *Oracle Application Server Reports Services Publishing Reports to the Web* for more information about dynamic environment switching

Locale Awareness in Oracle Application Server Discoverer

Oracle Application Server Discoverer can simultaneously support users with different locales. Discoverer always uses UTF-8 encoding for communication between the client and middle-tier services. Users may explicitly control the locale used for the user interface, or they may allow Oracle Application Server Discoverer to automatically determine a default. The order of precedence is:

1. Language and locale settings included in the URL for Oracle Application Server Discoverer
2. Language and locale settings specified in the Discoverer Connection (this is part of the Oracle Application Server Discoverer integration with Oracle Application Server Single Sign-On).
3. Language and locale setting specified in the user's browser
4. Language and locale of Oracle Application Server

For example, suppose a user's browser's language and locale are set to `German - Germany` and the user goes to the URL to start Oracle Application Server Discoverer. The HTML page returned to the user is displayed in German. If the user clicks on the Discoverer Connection, which has the language and locale specified as `English - US`, the Discoverer user interface appears in English. This is because the Discoverer Connection settings take precedence over the browser's settings.

3

Implementing HTML Features

This chapter contains the following topics:

- Implementing HTML Features for Global Applications
- Encoding HTML Pages
- Handling HTML Form Input
- Decoding HTTP Headers
- Encoding URLs
- Formatting HTML Pages to Accommodate Text in Different Languages
- Organizing the Content of HTML Pages for Translation

Implementing HTML Features for Global Applications

There are a variety of HTML features that you can use to enhance your global Internet applications. The following sections discuss some of the most important HTML features to keep in mind when designing your global applications.

Encoding HTML Pages

The encoding of an HTML page is important information for a browser and an Internet application. You can think of the **page encoding** as the character set used for the locale that an Internet application is serving. The browser needs to know about the page encoding so that it can use the correct fonts and character set mapping tables to display pages. Internet applications need to know about the HTML page encoding so they can process input data from an HTML form. To correctly specify the page encoding for HTML pages, Internet applications must:

- Choose a page encoding
- Encode HTML content in the desired encoding
- Correctly specify the HTML pages with the encoding name

Choosing an HTML Page Encoding for Monolingual Applications

The HTML page encoding is based on the user's locale. If the application is monolingual, it supports only one locale per instance. Therefore, you should encode HTML pages in the native encoding for that locale. The encoding should be equivalent to the Oracle character set specified by the `NLS_LANG` parameter in the Oracle HTTP Server configuration file.

Table 3–1 lists the Oracle character set names for the native encodings of the most commonly used locales, along with the corresponding Internet Assigned Numbers Authority (IANA) encoding names and Java encoding names. Use these character sets for monolingual applications.

> **See Also:** Chapter, "Setting NLS_LANG for a Monolingual Application Architecture"

Table 3–1 Native Encodings for Commonly Used Locales

Language	Oracle Character Set Name	IANA Encoding Name	Java Encoding Name
Western European	WE8MSWIN1252	ISO-8859-1	ISO8859_1
Central European	EE8MSWIN1250	ISO-8859-2	ISO8859_2
Japanese	JA16SJIS	Shift_JIS	MS932
Traditional Chinese	ZHT16MSWIN950	Big5	MS950
Simplified Chinese	ZHS16GBK	GB2312	GBK
Korean	KO16MSWIN949	EUC-KR	MS949
Arabic	AR8MSWIN1256	ISO-8859-6	ISO8859_6
Hebrew	IW8MSWIN1255	ISO-8859-8	ISO8859_8
Cyrillic	CL8MSWIN1251	ISO-8859-5	ISO8859_5
Baltic	BLT8MSWIN1257	ISO-8859-4	ISO8859_4
Greek	EL8MSWIN1253	ISO-8859-7	ISO8859_7
Thai	TH8TISASCII	TIS-620	TIS620

Table 3–1 Native Encodings for Commonly Used Locales (Cont.)

Language	Oracle Character Set Name	IANA Encoding Name	Java Encoding Name
Turkish	TR8MSWIN1254	ISO-8859-9	ISO8859_9
Universal	UTF8	UTF-8	UTF8

Choosing an HTML Page Encoding for Multilingual Applications

Multilingual applications need to determine the encoding used for the current user's locale at runtime and map the locale to the encoding as shown in Table 3–1.

Instead of using different native encodings for different locales, you can use UTF-8 for all page encodings. Using the UTF-8 encoding not only simplifies the coding for multilingual applications but also supports multilingual content. In fact, if a multilingual Internet application is written in Perl, the best choice for the HTML page encoding is UTF-8 because these programming environments do not provide an intuitive and efficient way to convert HTML content from UTF-8 to the native encodings of various locales.

There are limitations to using UTF-8 with the Netscape 4.x browser:

- HTTP multipart requests cannot contain non-ASCII file names.
- Localized versions of Windows NT 4.0 corrupt Asian characters in tool tips.
- Users must manually specify the font to be used for UTF-8 pages in the user preferences.

Netscape 6 resolves the second and third limitations.

Specifying the Page Encoding for HTML Pages

The best practice for monolingual and multilingual applications is to specify the encoding of HTML pages returned to the client browser. The encoding of HTML pages can tell the browser to:

- Switch to the specified encoding
- Return user input in the specified encoding

There are two ways to specify the encoding of an HTML page:

- Specifying the Encoding in the HTTP Header
- Specifying the Encoding in the HTML Page Header

If you use both methods, then specifying the encoding in the HTTP header takes precedence.

Specifying the Encoding in the HTTP Header

Include the Content-Type HTTP header in the HTTP specification. It specifies the content type and character set. The most commonly used browsers, such as Netscape 4.0 and Internet Explorer 4.0 or later, correctly interpret this header. The Content-Type HTTP header has the following form:

```
Content-Type: text/plain; charset=iso-8859-4
```

The `charset` parameter specifies the encoding for the HTML page. The possible values for the `charset` parameter are the IANA names for the character encodings that the browser supports. Table 3–1 shows commonly used IANA names.

Specifying the Encoding in the HTML Page Header

Use this method primarily for static HTML pages. Specify the character encoding in the HTML header as follows:

```
<meta http-equiv="Content-Type" content="text/html;charset=utf-8">
```

The `charset` parameter specifies the encoding for the HTML page. The possible values for the `charset` parameter are the IANA names for the character encodings that the browser supports. Table 3–1 shows commonly used IANA names.

Specifying the Page Encoding in Java Servlets and Java Server Pages

For both monolingual and multilingual applications, you can specify the encoding of an HTML page in the Content-Type HTTP header in a Java Server Page (JSP) using the `contentType` page directive. For example:

```
<%@ page contentType="text/html; charset=utf-8" %>
```

This is the MIME type and character encoding that the JSP file uses for the response it sends to the client. You can use any MIME type or IANA character set name that is valid for the JSP container. The default MIME type is `text/html`, and the default character set is ISO-8859-1. In the example, the character set is set to UTF-8. The character set of the `contentType` page directive directs the JSP engine to encode the dynamic HTML page and set the HTTP Content-Type header with the specified character set.

For Java Servlets, you can call the `setContentType()` method of the Servlet API to specify a page encoding in the HTTP header. The following `doGet()` function shows how you should call this method:

```
public void doGet(HttpServletRequest request, HttpServletResponse
response)throws ServletException, IOException
{

    // generate the MIME type and character set header
    response.setContentType("text/html; charset=utf-8");
    ...
    // generate the HTML page
    Printwriter out = response.getWriter();
    out.println("<HTML>");
    ...
    out.println("</HTML>");
}
```

You should call the `setContentType()` method before the `getWriter()` method because the `getWriter()` method initializes an output stream writer that uses the character set that the `setContentType()` method call specifies. Any HTML content written to the writer and eventually to a browser is encoded in the encoding that the `setContentType()` call specifies.

Specifying the Page Encoding in PL/SQL Server Pages

You can specify a page encoding for PL/SQL front-end applications and PL/SQL Server Pages (PSP) in two ways:

- Specify the page encoding in the `NLS_LANG` parameter in the corresponding DAD. Use this method for monolingual applications so that you can change the page encoding without changing the application code to support a different locale.

  See Also: Chapter, "Configuring Transfer Mode for mod_plsql Runtime" for more information about configuring DADs

- Specify the page encoding explicitly from within the PL/SQL procedures and PSPs. A page encoding that is specified explicitly overwrites the page encoding inherited from the `NLS_LANG` character set. Use this method for multilingual applications so that they can use different page encodings for different locales at runtime.

The specified page encoding tells the mod_plsql module and the Web Toolkit to tag the corresponding charset parameter in the Content-Type header of an HTML page and to convert the page content to the corresponding character set.

See Also: *PL/SQL User's Guide and Reference*

Specifying the Page Encoding in PL/SQL for Monolingual Environments

In order for monolingual applications to take the page encoding from the NLS_LANG parameter, the Content-Type HTTP header should not specify a page encoding. For PL/SQL procedures, the call to mime_header(), if any, should be similar to the following:

```
owa_util.mime_header('text/html',false);
```

For PSPs, the content type directive should be similar to the following:

```
<%@ page contentType="text/html"%>
```

Without the page encoding specified in the mime_header() function call or the content type directive, the Web Toolkit API uses the NLS_LANG character set as the page encoding by default, and converts HTML content to the NLS_LANG character set. Also, the Web Toolkit API automatically adds the default page encoding to the charset parameter of the Content-Type header.

Specifying the Page Encoding in PL/SQL for Multilingual Environments

You can specify a page encoding in a PSP the same way that you specify it in a JSP page. The following directive tells the PSP compiler to generate code to set the page encoding in the HTTP Content-Type header for this page:

```
<%@ page contentType="text/html; charset=utf-8" %>
```

To specify the encoding in the Content-Type HTTP header for PL/SQL procedures, use the Web Toolkit API in the PL/SQL procedures. The Web Toolkit API consists of the OWA_UTL package, which allows you to specify the Content-Type header as follows:

```
owa_util.mime_header('text/html', false, 'utf-8')
```

You should call the mime_header() function in the context of the HTTP header. It generates the following Content-Type header in the HTTP response:

```
Content-Type: text/html; charset=utf-8
```

After you specify a page encoding, the Web Toolkit API converts HTML content to the specified page encoding.

Specifying the Page Encoding in Perl

For Perl scripts running in the `mod_perl` environment, specify the encoding for an HTML page in the HTTP Content-Type header as follows:

```
$page_encoding = 'utf-8';
$r->content_type("text/html; charset=$page_encoding");
$r->send_http_header;
return OK if $r->header_only;
```

See Also: *Oracle HTTP Server Administrator's Guide*

Specifying the Page Encoding in Perl for Monolingual Applications

For monolingual applications, the encoding of an HTML page should be equivalent to:

- The character set used for the POSIX locale on which a Perl script runs
- The Oracle character set specified by the `NLS_LANG` parameter if the Perl script accesses the database

Specifying the Page Encoding in Perl for Multilingual Applications

For multilingual applications, Perl scripts should run in an environment where:

- Both the `NLS_LANG` character set and the character set used for the POSIX locale are equivalent to UTF-8
- The UTF8 Perl pragma is used

  This pragma tells the Perl interpreter to encode identifiers and strings in the UTF-8 encoding.

  See Also: *Oracle HTTP Server Administrator's Guide* for more information about the UTF-8 pragma

This environment allows the scripts to process data in any language in UTF-8. The page encoding of the dynamic HTML pages generated from the scripts, however, could be different from UTF-8. If so, then use the `UNICODE::MAPUTF8` Perl module to convert data from UTF-8 to the page encoding.

See Also: `http://www.cpan.org` to download the `UNICODE::MAPUTF8` Perl module

The following example illustrates how to use the `UNICODE::MAPUTF8` Perl module to generate HTML pages in the Shift_JIS encoding:

```
use Unicode::MapUTF8 qw(from_utf8)
# This shows how the UTF8 Perl pragma is specified
# but is NOT required by the from_utf8 function.
use utf8;
...
$page_encoding = 'Shift_JIS';
$r->content_type("text/html; charset=$page_encoding");
$r->send_http_header;
return OK if $r->header_only;
...
#html_lines contains HTML content in UTF-8
print (from_utf8({ -string=>$html_lines, -charset=>$page_encoding}));
...
```

The `from_utf8()` function converts dynamic HTML content from UTF-8 to the character set specified in the `charset` argument.

Specifying the Page Encoding in Oracle Application Server Mobile Services Applications

The page encoding for a Mobile Services application is specified in the application in the same way as other Java or JSP Internet applications. The page encoding specifies the encoding of the Mobile XML generated by the application, and it should be consistently specified in the Mobile XML prolog and the HTTP Content-Type header. The `HelloGlobe.jsp` application illustrates how the page encoding for the Mobile XML prolog should be specified.

Example 3–1 HelloGlobe.jsp

```
<?xml version="1.0" encoding="UTF-8"?>   (1)
<%@ page contentType="text/vnd.oracle.mobilexml; charset=UTF-8"%>   (2)
<SimpleResult>
   <SimpleContainer>
      <SimpleForm title="Hello Globe"
                  target="HelloGlobeReply.jsp" method="POST">
         <SimpleFormItem name="UserName" title="Your Name:" />
      </SimpleForm>
```

```
   </SimpleContainer>
</SimpleResult>
```

In this example, line (1) sets the content encoding XML prolog, and line (2) sets the content encoding in the HTTP Content-Type header.

Oracle Application Server Wireless converts the Mobile XML into the page encoding supported by the target device from the encoding information specified in the XML prolog and the HTTP Content-Type header. It then renders the content in the markup language supported by the target device. If the encodings specified in the XML prolog and the HTTP Content-Type header are inconsistent, the Oracle Application Server Wireless Mobile XML conversion will fail.

Specifying the Page Encoding in Oracle Application Server Web Cache Enabled Applications

When an edge side include (ESI) fragment is in a different page encoding from that of the corresponding ESI template, Oracle Application Server Web Cache converts the fragment to the page encoding of the template. This is to avoid cases where the content of a cached page is constructed in multiple page encodings. The character set conversion in Oracle Application Server Web Cache takes place only when both the template's and fragment's page encodings are known. Otherwise Oracle Application Server Web Cache assumes they are in the same page encoding, and therefore embeds the fragment into the template without converting the fragment.

Oracle Application Server Web Cache looks for the page encoding information only in the Content-Type header of an HTTP response. It does not look for the page encoding information within the content of the HTTP response.

To avoid losing information during the character set conversion of ESI fragments to ESI templates, applications should use a page encoding for ESI fragments that is a subset of the ESI template page encoding. There are two basic best practices for developers to consider:

1. Use UTF-8 as the page encoding for ESI templates, since UTF-8 is a superset of all other non-Unicode page encodings.
2. Use the same page encoding for ESI fragments and ESI templates. Character set conversion will not happen in this case.

Specifying the Page Encoding in Oracle Application Server Reports Services Applications

The page encodings that you use for different types of Reports Services applications depend on what type of report you are creating. This section discusses the page encoding options for Reports Services.

Specifying the Page Encoding in JSP Reports for the Web

You can specify the page encoding in JSP or HTML with the Web Source Editor in Reports Builder.

> **See Also:** "Specifying the Encoding in the HTML Page Header" and "Specifying the Page Encoding in Java Servlets and Java Server Pages" for more information.

Specifying the Page Encoding in HTML for Oracle Application Server Reports Services

Specify the HTML page encoding in the page header. For example, to specify a Japanese character set, include the following tag in the page header:

```
<META http-equiv="Content-Type" content="text/html;charset=SHIFT_JIS">
```

> **See Also:** "Specifying the Encoding in the HTML Page Header"

Reports Builder puts this tag in your report via the `Before Report Value` and `Before Form Value` properties. The default values for these properties are similar to the following:

```
<html><head><meta http-equiv="Content-Type" content="text/html;charset=&Encoding"></head>
```

The IANA locale name that is equivalent to the `NLS_LANG` setting for Oracle Application Server Reports Services is assigned to `&Encoding` dynamically at runtime. Thus you do not need to modify your report or Oracle Application Server Reports Services settings to include the proper locale.

> **See Also:** Reports Builder online help for more information

Specifying the Page Encoding in XML for Oracle Application Server Reports Services

Generally, when using XML, you would specify the encoding for XML by including a statement similar to the following as the Prolog at the first line in the outputted XML file:

```
<?xml version="1.0" encoding="SHIFT_JIS"?>
```

To set this Prolog in your report, you can specify the `XML Prolog Value` property of your report in or use the `SRW.SET_XML_PROLOG` built-in. The default value for the `XML Prolog Value` property is:

```
<?xml version="1.0" encoding="&Encoding"?>
```

In this case, Reports translates the value set as the `NLS_CHARACTERSET` into what is expected in the XML specification.

Note: You can overwrite the mapping by adding entries to your REPORTS_NLS_XML_CHARSET. The syntax is:

```
<old_name>=<new_name>[;<old_name>=<new_name>][;<old_name>=<new_
name>]...
```

Example:

```
ISO-8859-8=ISO-8859-8-1;CSEUCKR=EUC-KR;WINDOWS-949=EUC-KR;EUC-CN
=GBK;WINDOWS-936=GBK
```

See Also: Reports Builder online help for more information

Handling HTML Form Input

Applications generate HTML forms to get user input. For Netscape and Internet Explorer browsers, the encoding of the input always corresponds to the encoding of the forms for both POST and GET requests. In other words, if the encoding of a form is UTF-8, input text that the browser returns is encoded in UTF-8. Thus Internet applications can control the encoding of the form input by specifying the corresponding encoding in the HTML form that requests information.

How a browser passes input in a POST request is different from how it passes input in a GET request:

- For POST requests, the browser passes input as part of the request body. 8-bit data is allowed.
- For GET requests, the browser passes input as part of a URL as an embedded query string where every non-ASCII byte is encoded as `%XX`, where `XX` is the hexadecimal representation for the binary value of the byte.

HTML standards allow named and numbered entities. These special codes allow users to specify characters. For example, `&aelig;` and `&#230;` both refer to the character æ. Tables of these entities are available at

`http://www.w3.org/TR/REC-html40/sgml/entities.html`

Some browsers generate numbered or named entities for any input character that cannot be encoded in the encoding of an HTML form. For example, the Euro character and the character à (Unicode values 8364 and 224 respectively) cannot be encoded in Big5 encoding and are sent as `&#8364;` and `&agrave;` when the HTML encoding is Big5. However, the browser does not need to generate numbered or named entities if the page encoding of the HTML form is UTF-8 because all characters can be encoded in UTF-8. Internet applications that support page encoding other than UTF-8 need to be able to handle numbered and named entities.

Handling HTML Form Input in Java

In most JSP and Servlet containers, including Apache JServ, the Servlet API implementation assumes that incoming form input is in ISO-8859-1 encoding. As a result, when the`HttpServletRequest.getParameter()` API is called, all embedded `%XX` data in the input text is decoded, and the decoded input is converted from ISO-8859-1 to Unicode and returned as a Java string. The Java string returned is incorrect if the encoding of the HTML form is not ISO-8859-1. However, you can work around this problem by converting the form input data. When a JSP or Java Servlet receives form input in a Java string, it needs to convert it back to the original form in bytes, and then convert the original form to a Java string based on the correct encoding.

The following code converts a Java string to the correct encoding. The Java string `real` is initialized to store the correct characters from a UTF-8 form:

```
String original = request.getParameter("name");
try
{
    String real = new String(original.getBytes("8859_1"),"UTF8");
}
catch (UnsupportedEncodingException e)
```

```
{
    String real = original;
}
```

In addition to Java encoding names, you can use IANA encoding names as aliases in Java functions.

> **See Also:** Table 3–1 for mapping between commonly used IANA and Java encoding names

OC4J implements Servlet API 2.3, from which you can get the correct input by setting the `CharEncoding` attribute of the HTTP request object before calling the `getParameter()` function. Use the following code:

```
request.setCharacterEncoding("UTF8");
String real = request.getParameter("name");
```

Handling HTML Form Input in PL/SQL

The browser passes form input to PL/SQL procedures as PL/SQL procedure arguments. When a browser issues a POST or a GET request, it first sends the form input to the `mod_plsql` module in the encoding of the requesting HTML form. The `mod_plsql` module then decodes all `%XX` escape sequences in the input to their actual binary representations. It then passes the input to the PL/SQL procedure serving the request.

You should construct PL/SQL arguments you use to accept form input with the `VARCHAR2` datatype. Data in `VARCHAR2` are always encoded in the database character set. For example, the following PL/SQL procedure accepts two parameters in `VARCHAR2`:

```
procedure test(name VARCHAR2, gender VARCHAR2)
begin
...
end;
```

By default, the `mod_plsql` module assumes that the arguments of a PL/SQL procedure are in `VARCHAR2` datatype when it binds them. Using `VARCHAR2` as the argument datatype means that the module uses Oracle Character Set Conversion facility provided in Oracle Callable Library to convert form input data properly from the `NLS_LANG` character set, which is also your page encoding, to the database character set. The corresponding DAD specifies the `NLS_LANG` character set. As a result, the arguments passed as `VARCHAR2` should already be encoded in the database character set and be ready to use within the PL/SQL procedures.

Handling HTML Form Input in PL/SQL for Monolingual Applications

For monolingual application deployment, the NLS_LANG character set specified in the DAD is the same as the character set of the form input and the page encoding chosen for the locale. As a result, form input passed as VARCHAR2 arguments should be transparently converted to the database character set and ready for use.

Handling HTML Form Input in PL/SQL for Multilingual Applications

For multilingual application deployment, form input can be encoded in different character sets depending on the page encodings you choose for the corresponding locales. You can no longer use Oracle Character Set Conversion facility because the character set of the form input is not always the same as the NLS_LANG character set. Relying on this conversion corrupts the input. To resolve this problem, disable Oracle Character Set Conversion facility by specifying the same NLS_LANG character set in the corresponding DAD as the database character set. Once you disable the conversion, PL/SQL procedures receive form input as VARCHAR2 arguments. You must convert the arguments from the form input encoding to the database character set before using them. You can use the following code to convert the argument from ISO-8859-1 character set to UTF-8:

```
procedure test(name VARCHAR2, gender VARCHAR2)
begin
   name := CONVERT(name, 'AMERICAN_AMERICA.UTF8',
                  AMERICAN_AMERICA.WE8MSWIN1252')
   gender := CONVERT(gender, 'AMERICAN_AMERICA.UTF8',
                  AMERICAN_AMERICA.WE8MSWIN1252')
...
end;
```

See Also: Chapter, "Configuring the NLS_LANG Parameter"

Handling HTML Form Input in Perl

In the Oracle HTTP Server mod_perl environment, GET requests pass input to a Perl script differently than POST requests. It is good practice to handle both types of requests in the script. The following code gets the input value of the name parameter from an HTML form:

```
my $r = shift;
my %params = $r->method eq 'POST' ? $r->content : $r->args ;
my $name = $params{'name'} ;
```

For multilingual Perl scripts, the page encoding of an HTML form may be different from the UTF-8 encoding used in the Perl scripts. In this case, input data should be

converted from the page encoding to UTF-8 before being processed. The following example illustrates how the `Unicode::MapUTF8` Perl module converts strings from Shift_JIS to UTF-8:

```
use Unicode::MapUTF8 qw(to_utf8);
# This is to show how the UTF8 Perl pragma is specified,
# and is NOT required by the from_utf8 function.
use utf8;
...
my $page_encoding = 'Shift_JIS';
my $r = shift;
my %params = $r->method eq 'POST' ? $r->content : $r->args ;
my $name = to_utf8({-string=>$params{'name'}, -charset=>$page_encoding});
...
```

The `to_utf8()` function converts any input string from the specified encoding to UTF-8.

Handling Form Input in Oracle Application Server Mobile Services Applications

When a mobile service is registered to Oracle Application Server Wireless using the Wireless Tools administration tool, the Input Encoding parameter of the service must be specified. Oracle Application Server Wireless encodes URL parameters using the encoding specified in the Input Encoding parameter of the service. The mobile service application should be written so that it uses the same encoding as the Input Encoding parameter to interpret input from the target mobile devices. The `HelloGlobeReply.jsp` example illustrates how to handle the response from the service `HelloGlobe.jsp`, which is described in Example 3–1.

Example 3–2 HelloGlobeReply.jsp

```
<%xml version="1.0" encoding="UTF-8"?>
<%@ page contentType="text/vnd.oracle.mobilexml; charset=UTF-8"%>
<%
   request.setCharaecterEncoding("UTF-8");   (1)
   String name = request.getParameter("UserName");
%>
<SimpleResult>
   <SimpleContainer>
      <SimpleText>
         <SimpleTextItem>Hello <%=name%> !</SimpleTextItem>
      </SimpleText>
   </SimpleContainer>
</SimpleResult>
```

In this example, line (1) specifies that parameters are encoded using UTF-8.

This assumes that the Input Encoding parameter is specified as UTF-8 when the Master Service of `HelloGlobe.jsp` is created. The mobile service application should specify the same encoding for all input parameters that are received from the target device.

Decoding HTTP Headers

In all HTTP headers specific to Oracle Application Server, any value containing non-ASCII characters is MIME encoded according to the RFC 2047 specification. The encoded headers must be properly decoded before being used in an application. Applications deployed on Oracle Application Server may receive these HTTP headers.

Decoding HTTP Headers from Oracle Application Server Single Sign-On

When applications are using Oracle Application Server Single Sign-On (SSO) to authenticate a user, they need to decode the headers that SSO sends. The headers whose values may contain encoded non-ASCII characters include:

- `REMOTE_USER`
- `Osso-User-Dn`
- `Osso-Subscriber`
- `Osso-Subscriber-Dn`

For Java-based Web applications deployed on OC4J, the `REMOTE_USER` header is already interpreted for you in the `HTTPServletRequest.getRemoteUser()` method, and the `REMOTE_USER` header is removed from HTTP requests. For other types of Web applications, the `REMOTE_USER` header is present and should be properly decoded along with other headers. To decode a header value, you may use the `javax.mail.internet.MimeUtility` package of the Java Mail API. See the example in "Decoding String-type Mobile Context Information Headers in Oracle Application Server Wireless Services" for more details.

For PL/SQL applications, you need to write your own code to decode these header values.

Decoding String-type Mobile Context Information Headers in Oracle Application Server Wireless Services

String-type mobile context information, such as Login User Name (`X-Oracle-User.name`), User Display Name (`X-Oracle-User.DisplayName`), and Address Line of the Location (`X-Oracle.User.Location.AddressLine1`) are MIME encoded in the HTTP headers. Applications must decode them after they are retrieved from the HTTP request. For example, a JSP application may retrieve and decode the user's display name as follows:

```
<@ page import="java.io.*" %>
<@ page import="javax.mail.internet.MimeUtility" %>
<%
   String rawDisplayName = request.getHeader("X-Oracle-User.DisplayName");
   String displayName = null;
   try
   {
      displayName = MimeUtility.decodeText(rawDisplayName);
   }
   catch (UnsupportedEncodingException e)
   {
      // don't care
      displayName = rawDisplayName;
   }
%>
```

Encoding URLs

If HTML pages contain URLs with embedded query strings, you must escape any non-ASCII bytes in the query strings in the `%XX` format, where `XX` is the hexadecimal representation of the binary value of the byte. For example, if an Internet application embeds a URL that points to a UTF-8 JSP page containing the German name "Schloß," then the URL should be encoded as follows:

`http://`*`host.domain`*`/actionpage.jsp?name=Schlo%c3%9f`

Here, `c3` and `9f` represent the binary value in hexadecimal of the ß character in the UTF-8 encoding.

To encode a URL, be sure to complete the following tasks:

- Convert the URL into the encoding expected from the target object. This encoding is usually the same as the page encoding used in your application.
- Escape non-ASCII bytes of the URL into the `%XX` format.

Most programming environments provide APIs to encode and decode URLs. The following sections describe URL encoding in various environments:

- Encoding URLs in Java
- Encoding URLs in PL/SQL
- Encoding URLs in Perl

Encoding URLs in Java

If you construct a URL in a JSP or Java Servlet, you must escape all 8-bit bytes using their hexadecimal values prefixed by a percent sign. The `URLEncoder.encode(String s, String enc)` function provided in JDK 1.4 or above enables you to escape the URL in a given HTML page encoding. You need to specify the proper Java encoding name that corresponds to the page encoding in the second argument. See Table 3–1 for the Java encoding names of some commonly used page encodings.

If you are using JDK 1.1, 1.2, or 1.3, only the `URLEncoder.encode(String s)` function is available. It only encodes a URL in the Java default encoding. To make this function work for URLs in any encoding, you must add code to escape any non-ASCII characters in a URL into their hexadecimal representation, based on the encoding of your choice.

The following code shows an example of how to encode a URL based on the UTF-8 encoding:

```
String unreserved = new String("/\\-  _.!~*'()
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz 0123456789");
StringBuffer out = new StringBuffer(url.length());
for (int i = 0; i < url.length(); i++)
{
      int c = (int) url.charAt(i);
   if (unreserved.indexOf(c) != -1) {
        if (c == ' ') c = '+';
        out.append((char)c);
        continue;
   }
   byte [] ba;
   try {
        ba = url.substring(i, i+1).getBytes("UTF8");
   } catch (UnsupportedEncodingException e) {
        ba = url.getBytes();
   }
   for (int j=0; j < ba.length; j++)
```

```
   {
        out.append("%" + Long.toHexString((long)(ba[j]&0xff)).toUpperCase());
   }
}
String encodedUrl = out.toString();
```

Encoding URLs in PL/SQL

In Oracle9*i*, you can call the `ESCAPE()` function in the `UTL_URL` package to encode a URL in PL/SQL. You can call the `ESCAPE()` function as follows:

```
encodedURL varchar2(100);
url varchar2(100);
charset varchar2(40);
...
encodedURL := UTL_URL.ESCAPE(url, FALSE, charset);
```

The `url` argument is the URL that you want to encode. The `charset` argument specifies the character encoding used for the encoded URL. Use a valid Oracle character set name for the `charset` argument. To encode a URL in the database character set, always specify the `charset` argument as NULL.

> **See Also:** Table 3–1 for a list of commonly used Oracle character set names

Encoding URLs in Perl

You can encode a URL in Perl by using the `escape_uri()` function of the `Apache::Util` module as follows:

```
use Apache::Util qw(escape_uri);
...
$escaped_url   = escape_uri( $url );
...
```

The `escape_uri()` function takes the bytes from the `$url` input argument and encodes them into the `%XX` format. If you want to encode a URL in a different character encoding, you need to convert the URL to the target encoding before calling the `escape_uri()` function. Perl provides some modules for character conversion.

> **See Also:** `http://www.cpan.org` for Perl character conversion modules

Formatting HTML Pages to Accommodate Text in Different Languages

Design the format of HTML pages according to the following guidelines:

- Allow table cells to resize themselves as the enclosed text expands, instead of hard-coding the widths of the cells. The following is an example of hard-coding the width of a cell:

  ```
  <TD WIDTH="50">
  ```

  If you must specify the widths of cells, then externalize the width values so that translators can adjust them with the translated text.

- Do not specify fonts directly in the HTML pages because they may not contain glyphs for all languages that the application supports. Instead, each element should inherit from a class in a cascading style sheet (CSS) that specifies fonts and font sizes.

- For bidirectional languages such as Arabic and Hebrew, the pages should have a `DIR` attribute in the `<HTML>` tag to indicate that the direction of the language displayed is from right to left. The `<HTML DIR="RTL">` tag causes all components of an HTML page to follow the direction of the HTML tag. To make direction settings seamless to developers, set the direction in the CSS file as follows:

  ```
  HTML{ direction:rtl }
  ```

  CSS level 2 introduced the direction property, which is supported in Internet Explorer 5.0.

- Text alignment should be sensitive to the direction of the text. In HTML, LEFT and RIGHT are absolute alignments. When the direction of the text is from left to right, as in English, the alignment should be LEFT. When the direction of the text is from right to left, as in Hebrew, the alignment should be RIGHT.

- It is good practice to provide Cascading Style Sheets (CSS) for different locales or groups of locales and use them to control HTML page rendering. Using a CSS isolates the locale-specific formatting information from HTML pages. Applications should dynamically generate CSS references in HTML pages corresponding to the user's locale so that the pages can be rendered with the corresponding locale-specific formats. Locale-specific information in the CSS file should include:

  - Font names and sizes
  - Alignments (for bidirectional language support only)

- Direction of text (for bidirectional language support only)

Organizing the Content of HTML Pages for Translation

You should have the user interface (UI) and content presented in HTML pages translated. Translatable sources for the content of an HTML page belong to the following categories:

- Static files such as HTML, images, and cascading style sheets (CSS)
- Static UI strings stored as Java resource bundles used by Java Servlets and JSPs
- Static UI strings stored as POSIX message files used by C/C++ programs and Perl scripts
- Static UI strings stored as relational data in a database used by PL/SQL procedures and PL/SQL Server Pages
- Dynamic content such as product information stored in the database

This section contains the following topics:

- Translation Guidelines for HTML Page Content
- Organizing Static Files for Translation
- Organizing Translatable Static Strings for Java Servlets and Java Server Pages
- Organizing Translatable Static Strings in C/C++ and Perl
- Organizing Translatable Static Strings in Message Tables
- Organizing Translatable Dynamic Content in Application Schema

Translation Guidelines for HTML Page Content

When creating translatable content, developers should follow these translation guidelines:

- Externalize to resource files all static and translatable UI strings used in programs such as Java Servlets, Java Server Pages, Perl scripts, PL/SQL procedures, and PL/SQL Server Pages. These resource files can then be translated independent of program code.
- All dynamic text in an HTML page must be able to expand by at least 30% without overlapping adjacent objects to allow for text expansion that can result from translation. The HTML page should look acceptable after expanding strings by 30%.

- Avoid concatenating strings to form sentences at runtime. The concatenated translated strings might not have the same meaning as the original strings. Use the string formatting functions provided by different programming languages to substitute runtime values for placeholders.
- Avoid embedding text into images and graphics because they are often not easy to translate.
- JavaScript code must not include any translatable strings. JavaScript is hard to translate. Instead, applications should externalize translatable strings, if any, into resource files or message tables. Applications should construct JavaScript code at runtime and replace the dynamic text with text corresponding to the user's locale.
- Because translations are often not available in the initial release of an application, it is important to make the application work when the corresponding translation is not available by putting a fallback mechanism in the application. The fallback mechanism can be as simple as using English information or as complex as using the closest language available. For example, the `fr-CA` locale is French (Canadian). The fallback for this language can be `fr` (French) or `en` (English). A simple way to find the closest possible language is to remove the territory part of the ISO locale name. It is up to the application how the fallback mechanism behaves.

Organizing Static Files for Translation

You should organize translatable HTML, images, and CSS files into different directories from non-translatable static files so that you can zip files under the locale-specific directory for translation. There are many possible ways to define the directory structure to hold these files. For example:

```
/docroot/images         - Non-translatable images
/docroot/html           - HTML common to all languages
/docroot/css            - Style sheets common to all languages
/docroot/<lang>         - Locale directory such as en, fr, ja etc.
/docroot/<lang>/images  - Images specific for <lang>
/docroot/<lang>/html    - HTMLs specific for <lang>
/docroot/<lang>/css     - Style sheets specific for <lang>
```

You can replace the `<lang>` placeholder with the ISO locale names. Based on the above structure, you must write a utility function called `getLocalizedURL()` to take a URL as a parameter and look for the available language file from this structure. Whenever you reference an HTML, image, or CSS file in an HTML page, the Internet application should call this function to construct the path of the

translated file corresponding to the current locale and fall back appropriately if the translation does not exist. For example, if the path `/docroot/html/welcome.html` is passed to the `getLocalizedURL()` function and the current locale is `fr_CA`, then the function looks for the following files in the order shown:

```
/docroot/fr_CA/html/welcome.html
/docroot/fr/html/welcome.html
/docroot/en/html/welcome.html
/docroot/html/welcome.html
```

The function returns the first file that exists. This function always reverts to English when the translated version corresponding to the current locale does not exist.

For Internet applications that use UTF-8 as the page encoding, the encoding of the static HTML files should also be UTF-8. However, translators usually encode translated HTML files in the native encoding of the target language. To convert the translated HTML into UTF-8, you can use the JDK `native2ascii` utility shipped with Oracle Application Server.

For example, to convert a Japanese HTML file encoded in Shift_JIS into UTF-8:

1. Replace the value of the `charset` parameter in the Content-Type HTML header in the `<meta>` tag with UTF-8.
2. Use the `native2ascii` utility to copy the Japanese HTML file to a new file called `japanese.unicode`:

   ```
   native2ascii -encoding MS932 japanese.html japanese.unicode
   ```

3. Use the `native2ascii` utility to convert the new file to Unicode:

   ```
   native2ascii -reverse -encoding UTF8 japanese.unicode japanese.html
   ```

See Also:

- *Oracle9i SQLJ Developer's Guide and Reference* in the Oracle Database Documentation Library
- JDK documentation at `http://java.sun.com`

for more information about the `native2ascii` utility

Organizing Translatable Static Strings for Java Servlets and Java Server Pages

You should externalize translatable strings within Java Servlets and JSPs into Java resource bundles so that these resource bundles can be translated independent of

the Java code. After translation, the resource bundles carry the same base class names as the English bundles, but with the Java locale name as the suffix. You should place the bundles in the same directory as the English resource bundles for the Java resource bundle look-up mechanism to function properly.

> **See Also:** JDK documentation at `http://java.sun.com` for more information about Java resource bundles

Some people may disagree about externalizing JSP strings to resource bundles because it seems to defeat the purpose of using JSPs. There are two reasons for externalizing JSPs strings:

- Translating JSPs is error-prone because they consist of Java code that is not familiar to translators
- The translation process should be separated from the development process so that translation can take place in parallel to development on JSPs. This eliminates the huge effort of merging the translated JSPs with the most up-to-date JSPs that contain bug fixes to the embedded Java code.

You can use resource bundles in your Java programs by providing your own subclass of the `ResourceBundle` class. Additionally, Java provides two subclasses of the `ResourceBundle` abstract class: `ListResourceBundle` and `PropertyResourceBundle`. It is good practice to provide your implementation of the `ResourceBundle` class as a subclass of `ListResourceBundle`. The main reasons are:

- List resource bundles are essentially Java programs that must be compiled. Translation errors can be caught at compile time. Property resource bundles are text files read directly from Java. Translation errors can only be caught at runtime.
- Property resource bundles expose all string data in your Internet application to users. There are potential security and support issues for your application.

The following is an example of a list resource bundle:

```
import java.util.ListResourceBundle;
public class Resource extends ListResourceBundle {
    public Object[][] getContents() {
        return contents;
    }
    static final Object[][] contents =
    {
       {"hello", "Hello World"},
```

```
        ...

    };
}
```

Translators usually translate list resource bundles in the native encoding of the target language. Japanese list resource bundles encoded in Shift_JIS cannot be compiled on an English system because the Java compiler expects source files that are encoded in ISO-8859-1. In order to build translated list resource bundles in a platform-independent manner, you need to run the JDK `native2ascii` utility to escape all non-ASCII characters to Unicode escape sequences in the `\uXXXX` format, where `XXXX` is the Unicode value in hexadecimal. For example:

```
native2ascii -encoding MS932 resource_ja.java resource_ja.tmp
```

Java provides a default fallback mechanism for resource bundles when translated resource bundles are not available. An application only needs to make sure that a base resource bundle without any locale suffix always exists in the same directory. The base resource bundle should contains strings in the fallback language. As an example, Java looks for a resource bundle in the following order when the `fr_CA` Java locale is specified to the `getBundle()` function:

```
resource_fr_CA
resource_fr
resource_en_US /* where en_US is the default Java locale */
resource_en
resource (base resource bundle)
```

Retrieving Strings in Monolingual Applications

At runtime, monolingual applications can get strings from a resource bundle of the default Java locale as follows:

```
ResourceBundle rb = ResourceBundle.getBundle("resource");
String helloStr = rb.getString("hello");
```

Retrieving Strings in Multilingual Applications

Because the user's locale is not fixed in multilingual applications, they should call the `getBundle()` method by explicitly specifying a Java locale object that corresponds to the user's locale. The Java locale object is called `user_locale` in the following example:

```
ResourceBundle rb = ResourceBundle.getBundle("resource", user_locale);
String helloStr = rb.getString("hello");
```

Organizing Translatable Static Strings in C/C++ and Perl

For C/C++ programs and Perl scripts running on UNIX platforms, externalize static strings in C/C++ or Perl scripts to POSIX message files. For programs running on Windows platforms, externalize static strings to message tables in a database because Windows does not support POSIX message files.

> **See Also:** "Organizing Translatable Static Strings in Message Tables"

Message files (with the `.po` file extension) associated with a POSIX locale are identified by their domain names. You need to compile them into binary objects (with the `.mo` file extension) and place them into the directory corresponding to the POSIX locale. The path name for the POSIX locale is implementation-specific. For example, the Solaris `msgfmt` utility compiles a Canadian French message file, `resource.po`, and places it into the `/usr/lib/locale/fr_CA/LC_MESSAGES` directory on Solaris.

> **See Also:** Operating system documentation for `gettext`, `msgfmt`, and `xgettext`

The following is an example of a `resource.po` message file:

```
domain "resource"
msgid "hello"
msgstr "Hello World"
...
```

Note that the encoding used for the message files must match the encoding used for the corresponding POSIX locale.

Instead of putting binary message files into an implementation-specific directory, you should put them into an application-specific directory and use the `binddomain()` function to associate a domain with a directory. The following piece of Perl script uses the `Locale::gettext` Perl module to get a string from a POSIX message file:

```
use Locale::gettext;
use POSIX;
...
setlocale( LC_ALL, "fr_CA" );
textdomain( "resource" );
binddomain( "resource", "/usr/local/share");
print gettext( "hello" );
```

The domain name for the resource file is `resource`, the ID of the string to be retrieved is `hello`, the translation to be used is Canadian French (`fr_ca`), and the directory for the `binary.mo` files is `/usr/local/share/fr_CA/LC_MESSAGES`.

> **See Also:** `http://www.cpan.org` to download the `Locale:gettext` Perl module

Organizing Translatable Static Strings in Message Tables

Message tables mainly store static translatable strings used by PL/SQL procedures and PSPs. You can also use them for some C/C++ programs and Perl scripts. The tables should have a language column to identify the language of static strings so that accessing applications can retrieve messages based on the user's locale. The table structure should be similar to the one below:

```
CREATE TABLE messages
( msgid   NUMBER(5)
, langid  VARCHAR2(10)
, message VARCHAR2(4000)
);
```

The primary key for this table consists of the `msgid` and `langid` columns. One good choice for the values in these columns is the Oracle language abbreviations of corresponding locales. Using the Oracle language abbreviation allows applications to retrieve translated information transparently by issuing a query on the message table.

> **See Also:** *Oracle9i Globalization Support Guide* in the Oracle Database Documentation Library for a list of Oracle language abbreviations

To provide a fallback mechanism when the translation of a message is not available, create the following views on top of the message table defined in the previous example:

```
-- fallback language is English which is abbreviated as 'US'.
CREATE VIEW default_message_view AS
 SELECT msgid, message
 FROM messages
 WHERE langid = 'US';
/
-- create view for services, with fall-back mechanism
CREATE VIEW messages_view AS
```

```
SELECT d.msgid,
       CASE WHEN t.message IS NOT NULL
            THEN t.message
            ELSE d.message
       END AS message
FROM default_view d,
     translation  t
WHERE t.msgid (+) = d.msgid AND
      t.langid (+) = sys_context('USERENV', 'LANG');
```

Messages should be retrieved from the `messages_view` view that provides the logic to provide a fallback message in English by joining the `default_message_view` view with the `messages` table. The `sys_context()` SQL function returns the Oracle language abbreviation of the locale for the current database session. This locale should be initialized to the user's locale at the time when the session is created.

To retrieve a message, an application should use the following query:

```
SELECT message FROM message_view WHERE msgid = 'hello';
```

The `NLS_LANGUAGE` parameter of a database session defines the language of the message that the query retrieves. Note that there is no language information needed for the query with this message table schema.

In order to minimize the load to the database, you should set up all message tables and their associated views on an Oracle Application Server instance as a front end to the database where PL/SQL procedures and PSPs run.

Organizing Translatable Dynamic Content in Application Schema

An application schema stores translatable dynamic information that the application uses, such as product names and product descriptions. The following shows an example of a table that stores all the products of an Internet store. The translatable information for the table is the product description and the product name.

```
CREATE TABLE product_information
    ( product_id          NUMBER(6)
    , product_name        VARCHAR2(50)
    , product_description VARCHAR2(2000)
    , category_id         NUMBER(2)
    , warranty_period     INTERVAL YEAR TO MONTH
    , supplier_id         NUMBER(6)
    , product_status      VARCHAR2(20)
    , list_price          NUMBER(8,2)
```

```
    );
```

To store product names and product descriptions in different languages, create the following table so that the primary key consists of the `product_id` and `language_id` columns:

```
CREATE TABLE product_descriptions
    ( product_id             NUMBER(6)
    , language_id            VARCHAR2(3)
    , translated_name        NVARCHAR2(50)
    , translated_description NVARCHAR2(2000)
    );
```

Create a view on top of the tables to provide fallback when information is not available in the language that the user requests. For example:

```
CREATE VIEW product AS
SELECT i.product_id
,      d.language_id
,      CASE WHEN d.language_id IS NOT NULL
            THEN d.translated_name
            ELSE i.product_name
       END    AS product_name
,      i.category_id
,      CASE WHEN d.language_id IS NOT NULL
            THEN d.translated_description
            ELSE i.product_description
       END    AS product_description
,      i.warranty_period
,      i.supplier_id
,      i.product_status
,      i.list_price
FROM   product_information  i
,      product_descriptions d
WHERE  d.product_id  (+) = i.product_id
AND    d.language_id (+) = sys_context('USERENV','LANG');
```

This view performs an outer join on the `product_information` and `production_description` tables and selects the rows with the `language_id` equal to the Oracle language abbreviation of the current database session.

To retrieve a product name and product description from the product view, an application should use the following query:

```
SELECT product_name, product_description FROM product
      WHERE product_id = '1234';
```

This query retrieves the translated product name and production description corresponding to the value of the NLS_LANGUAGE session parameter. Note that you do not need to specify any language information in the query.

4

Using a Centralized Database

This chapter contains the following topics:

- Using a Centralized Database and Accessing the Database Server
- Using JDBC to Access the Database
- Using PL/SQL to Access the Database
- Using Perl to Access the Database
- Using C/C++ to Access the Database

Using a Centralized Database and Accessing the Database Server

A centralized Unicode database is a feature of both the monolingual approach and the multilingual approach to developing globalized Internet applications. Using a centralized database has the following advantages:

- It provides a complete view of your data. For example, you can query for the number of customers worldwide or the worldwide inventory level of a product.
- It is easier to manage a centralized database than several distributed databases.

The database character set should be Unicode. You can use Unicode to store and manipulate data in several languages. Unicode is a universal character set that defines characters in almost all languages in the world. Oracle9*i* databases can store Unicode data in one of the following encoding forms:

- UTF-8: Each character is 1 to 4 bytes long.
- UTF-16: Each character is either 2 or 4 bytes long.
- UTF-32: Each character is 4 bytes long.

See Also:

- Chapter , "Configuring Oracle HTTP Server and OC4J for Global Deployment"
- *Oracle9i Globalization Support Guide* in the Oracle Database Documentation Library

There are several methods by which Internet applications can access the database server through Oracle Application Server. Any Java-based Internet applications that use technologies such as Java Servlets, JSPs, and EJBs can use the Oracle JDBC drivers for database connectivity.

Because Java strings are always Unicode-encoded, JDBC transparently converts text data from the database character set to Unicode and vice versa. Java Servlets and JSPs that interact with an Oracle database should ensure the following:

- The Java strings returned from the database are converted to the encoding of the HTML page being constructed
- Form inputs are converted from the encoding of the HTML form to Unicode before being used in calling the JDBC driver

For non-Java Internet applications that use programming technologies such as Perl, PL/SQL, and C/C++, text data retrieved from or inserted into a database are encoded in the character set specified by the `NLS_LANG` parameter. The character set used for the POSIX locale should match the `NLS_LANG` character set so that data from the database can be directly processed with the POSIX locale-sensitive functions in the applications.

See Also: Chapter 5, "Configuring Oracle Application Server for Global Deployment"

For multilingual applications, the `NLS_LANG` character set and the page encoding should both be UTF-8 to avoid character set conversion and possible data loss.

Using JDBC to Access the Database

Use the Oracle JDBC drivers provided in Oracle Application Server for Oracle9*i* database access when you use JSPs and Java Servlets. Oracle Application Server provides two client-side JDBC drivers that you can deploy with middle-tier applications:

- JDBC OCI driver, which requires the Oracle client library

- JDBC Thin driver, which is a pure Java driver

Oracle JDBC drivers transparently convert character data from the database character set to Unicode for the SQL CHAR data types and the SQL NCHAR data types. As a result of this transparent conversion, JSPs and Java Servlets calling Oracle JDBC drivers can bind and define database columns with Java strings and fetch data into Java strings from the result set of a SQL execution.

You can use a Java string to bind the NAME and ADDRESS columns of a customer table. Define the columns as VARCHAR2 and NVARCHAR2 data types, respectively. For example:

```
String cname = request.getParameter("cname")
String caddr = request.getParameter("caddress");
OraclePreparedStatement pstmt = conn.prepareStatement("insert into" +
        "CUSTOMERS (NAME, ADRESS) values (?, ?) ");
pstmt.setString(1, cname);
pstmt.setFormOfUse(2, OraclePreparedStatement.FORM_NCHAR);
pstmt.setString(2, caddr);
pstmt.execute();
```

To bind a Java string variable to the ADDRESS column defined as NVARCHAR2, you should call the setFormOfUse() method before the setString() method.

> **See Also:** *Oracle9i JDBC Developer's Guide and Reference* in the Oracle Database Documentation Library

The Oracle JDBC drivers set the values for the NLS_LANGUAGE and NLS_TERRITORY session parameters to the values corresponding to the default Java locale when the database session was initialized. For monolingual applications, the Java default locale is configured to match the user's locale. Hence the database connection is always synchronized with the user's locale.

Using PL/SQL to Access the Database

PL/SQL procedures and PSPs use SQL to access data in the local Oracle9*i* database. They can also use SQL and database links to access data from a remote Oracle9*i* database.

For example, you can call the following PL/SQL procedure from the mod_plsql module. It inserts a record into a customer table with the customer name column defined as VARCHAR2 and the customer address column defined as NVARCHAR2:

```
procedure addcustomer( cname varchar2 default NULL, caddress nvarchar2 default
```

```
NULL) is
begin
   ....
   if (cname is not null) then
        caddr :=TO_NCHAR(address);
        insert into customers (name, address) values (cname, caddr);
        commit;
   end if;
end;
```

Note that Apache `mod_plsql` does not support `NVARCHAR` argument passing. As a result, PL/SQL procedures have to use `VARCHAR2` for arguments and convert them to `NVARCHAR` explicitly before executing the `INSERT` statement.

The example uses static SQL to access the customer table. You can also use the `DBMS_SQL` PL/SQL package to access data in the database, using dynamic SQL.

> **See Also:** *Oracle9i Supplied PL/SQL Packages Reference* in the Oracle Database Documentation Library

Using Perl to Access the Database

Perl scripts access Oracle9*i* databases using the DBI/DBD driver for Oracle. The DBI/DBD driver is part of Oracle Application Server. It calls Oracle Callable Interface (OCI) to access the databases. The data retrieved from or inserted into the databases is encoded in the `NLS_LANG` character set. Perl scripts should do the following:

- Initialize a POSIX locale with the locale specified in the `LC_ALL` environment variable
- Use a character set equivalent to the `NLS_LANG` character set

This allows you to process data retrieved from the databases with POSIX string manipulation functions.

The following code shows how to insert a row into a customer table in an Oracle9*i* database through the DBI/DBD driver.

```
Use Apache::DBI;
...
# Connect to the database
$constr = 'host=dlsun1304.us.oracle.com;sid=icachedb;port=1521' ;
$usr = 'system' ;
$pwd = 'manager' ;
$dbh = DBI->connect("dbi:Oracle:$constr", $usr, $pwd, {AutoCommit=>1} ) ||
```

```
        $r->print("Failed to connect to Oracle: " . DBI->errstr );

# prepare the statement
$sql = 'insert into customers (name, address) values (:n, :a)';
$sth = $dbh->prepare( $sql );
$sth->bind_param(':n' , $cname);
$sth->bind_param(':a', $caddress);
$sth->execute();
$sth->finish();
$dbh->disconnect();
```

If the target columns are of the SQL NCHAR data types, then you need to specify the form of use flag for each bind variable. For example, if the address column is of NVARCHAR2 datatye, you need to add the `$sth->func()` function call before executing the SQL statement:

```
use DBD::Oracle qw(:ora_forms);
...
$sql = 'insert into customers (name, address) values (:n, :a)';
$sth = $dbh->prepare($sql);
$sth->bind_param(':n', $cname);
$sth->bind_param(':a', $caddress);
$sth->func( { ':a' => ORA_NCHAR }, 'set_form');
$sth->execute();
$sth->finish();
$dbh->disconnect();
```

To properly process UTF-8 data in a multilingual application, Perl scripts should do the following:

- Use a POSIX locale associated with the UTF-8 character set
- Use the UTF-8 Perl module to indicate that all strings in the Perl scripts are in UTF-8

Using C/C++ to Access the Database

C/C++ applications access the Oracle9*i* databases with OCI or Pro*C/C++. You can call OCI directly or use the Pro*C/C++ interface to retrieve and store Unicode data in a UTF-8 database and in SQL NCHAR data types.

Generally, data retrieved from and inserted into the database is encoded in the NLS_LANG character set. C/C++ programs should use the same character set for their POSIX locale as the NLS_LANG character set. Otherwise, the POSIX string functions cannot be used on the character data retrieved from the database, and the

character data encoded in the POSIX locale may be corrupted when it is inserted into the database.

For multilingual applications, you may want to use the Unicode API provided in the OCI library instead of relying on the NLS_LANG character set. This alternative is good for applications written for platforms such as Windows NT/2000, which implement the wchar_t datatype using UTF-16 Unicode. Using the Unicode API for those platforms bypasses some unnecessary data conversions that using the regular OCI API requires.

This section includes the following topics:

- Using the OCI API to Access the Database
- Using the Unicode API Provided with OCI to Access the Database
- Using Unicode Bind and Define in Pro*C/C++ to Access the Database

Note: OCI libraries are part of Oracle Application Server. You do not need to install the Oracle9*i* database client to use them.

Using the OCI API to Access the Database

This example shows how to bind and define the VARCHAR2 and NVARCHAR2 columns of a customer table in C/C++. It uses OCI and is based on the NLS_LANG character set. Note that the text datatype is a macro for unsigned char.

```
text *sqlstmt= (text *)"SELECT name, address FROM customers
                        WHERE id = :cusid";
text cname[100];                        /* Customer Name */
text caddr[200];                        /* Customer Address */
text custid[10] = "9876";               /* Customer ID */
ub2 cform = SQLCS_NCHAR;                /* Form of Use for NCHAR types */
...
OCIStmtPrepare (stmthp, errhp, sqlstmt,
                (ub4)strlen ((char *)sqlstmt),
                (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));
/* Bind the custid buffer */
OCIBindByName(stmthp, &bnd1p, errhp, (text*)":custid",
              (sb4)strlen((char *)":custid"),
              (dvoid *) custid, sizeof(cust_id), SQLT_STR,
              (dvoid *)&insname_ind, (ub2 *) 0, (ub2 *) 0,
              (ub4) 0,(ub4 *)0, OCI_DEFAULT);

/* Define the cname buffer for VARCHAR */
```

```
OCIDefineByPos (stmthp, &dfn1p, errhp, (ub4)1, (dvoid *)cname,
                (sb4)sizeof(cname), SQLT_STR,
                (dvoid *)0, (ub2 *)0, (ub2 *)0, (ub4)OCI_DEFAULT);

/* Define the caddr buffer for the address column in NVARCHAR2 */
OCIDefineByPos (stmthp, &dfn2p, errhp, (ub4)2, (dvoid *)caddr,
                (sb4)sizeof(caddr), SQLT_STR,
                (dvoid *)0, (ub2 *)0, (ub2 *)0, (ub4)OCI_DEFAULT);
OCIAttrSet((void *) dfn2p, (ub4) OCI_HTYPE_DEFINE, (void *) &cform, (ub4) 0,
           (ub4)OCI_ATTR_CHARSET_FORM, errhp);
...
```

Using the Unicode API Provided with OCI to Access the Database

You can use the Unicode API that the OCI library provides for multilingual applications.

Turn on the Unicode API by specifying Unicode mode when you create an OCI environment handle. Any handle inherited from the OCI environment handle is set to Unicode mode automatically. By changing to Unicode mode, all text data arguments to the OCI functions are assumed to be in the Unicode text (`utext*`) datatype and in UTF-16 encoding. For binding and defining, the data buffers are assumed to be `utext` buffers in UTF-16 encoding.

The program code for the Unicode API is similar to the code for the non-Unicode OCI API, with the following exceptions:

- All `text` data types are changed to the `utext` datatype, which is a macro of the unsigned short datatype
- All literal strings are changed to Unicode literal strings
- All `strlen()` functions are changed to `wcslen()` functions to calculate the string length in number of Unicode characters instead of bytes

The following Windows program shows how to do these tasks:

- Create an OCI environment handle with Unicode mode turned on
- Bind and define the name column in `VARCHAR2` and the address column in `NVARCHAR2` of the `customers` table

```
utext *sqlstmt= (text *)L"SELECT name, address FROM customers
                         WHERE id = :cusid";
utext cname[100];                       /* Customer Name */
utext caddr[200];                       /* Customer Address */
text custid[10] = "9876";               /* Customer ID */
```

```
ub1 cform = SQLCS_NCHAR;              /* Form of Use for NCHAR types */
...
/* Use Unicode OCI API by specifying UTF-16 mode */
status = OCIEnvCreate((OCIEnv **)&envhp, OCI_UTF16, (void *)0,
                      (void *(*) ()) 0, (void *(*) ()) 0, (void(*) ()) 0,
                      (size_t) 0, (void **)0);
...
OCIStmtPrepare (stmthp, errhp, sqlstmt,
                (ub4)wcslen ((char *)sqlstmt),
                (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));
/* Bind the custid buffer */
OCIBindByName(stmthp, &bnd1p, errhp, (constant text*) L":custid",
              (sb4)wcslen(L":custid"),
              (dvoid *) custid, sizeof(cust_id), SQLT_STR,
              (dvoid *)&insname_ind, (ub2 *) 0, (ub2 *) 0,
              (ub4) 0,(ub4 *)0, OCI_DEFAULT);

/* Define the cname buffer for the name column in VARCHAR2 */
OCIDefineByPos (stmthp, &dfn1p, errhp, (ub4)1, (dvoid *)cname,
                (sb4)sizeof(cname), SQLT_STR,
                (dvoid *)0, (ub2 *)0, (ub2 *)0, (ub4)OCI_DEFAULT);

/* Define the caddr buffer for the address column in NVARCHAR2 */
OCIDefineByPos (stmthp, &dfn2p, errhp, (ub4)2, (dvoid *)caddr,
                (sb4)sizeof(caddr), SQLT_STR,
                (dvoid *)0, (ub2 *)0, (ub2 *)0, (ub4)OCI_DEFAULT);
OCIAttrSet((void *) dfn2p, (ub4) OCI_HTYPE_DEFINE, (void *) &cform, (ub4) 0,
           (ub4)OCI_ATTR_CHARSET_FORM, errhp);
...
```

Using Unicode Bind and Define in Pro*C/C++ to Access the Database

You can use Unicode bind and define in Pro*C/C++ for multilingual applications.

Pro*C/C++ lets you specify UTF-16 Unicode buffers for bind and define operations. There are two ways to specify UTF-16 buffers in Pro*C/C++:

- Use the `utext` datatype, which is an alias for the unsigned short datatype in C/C++
- Use the `uvarchar` datatype provided by Pro*C/C++. It will be preprocessed to a `struct` with a length field and a `utext` buffer field.

```
struct uvarchar
{
   ub2 len;         /* length of arr */
```

```
   utext arr[1] ;   /* UTF-16 buffer */
};
typedef struct uvarchar uvarchar ;
```

In the following example, there are two host variables: `cname` and `caddr`. The `cname` host variable is declared as a `utext` buffer containing 100 UTF-16 code units (unsigned short) for the customer name column in the `VARCHAR2` datatype. The `caddr` host variable is declared as a `uvarchar` buffer containing 50 UCS2 characters for the customer address column in the `NVARCHAR2` datatype. The `len` and `arr` fields are accessible as fields of a `struct`.

```
#include <sqlca.h>
#include <sqlucs2.h>

main()
{
   ...
   /* Change to STRING datatype:    */
   EXEC ORACLE OPTION (CHAR_MAP=STRING) ;
   utext cname[100] ;              /* unsigned short type */
   uvarchar caddr[200] ;           /* Pro*C/C++ uvarchar type */
   ...
   EXEC SQL SELECT name, address INTO :cname, :caddr FROM customers;
   /* cname is NULL-terminated */
   wprintf(L"ENAME = %s, ADDRESS = %.*s\n", cname, caddr.len, caddr.arr);
   ...
}
```

Part II

Deployment

Part II contains the following chapters:

- Chapter 5, "Configuring Oracle Application Server for Global Deployment"
- Chapter 6, "A Multilingual Demo for Oracle Application Server"

5

Configuring Oracle Application Server for Global Deployment

When developing and deploying global Internet applications with Oracle Application Server, you need to consider the following tasks:

- Installing Oracle Application Server for Global Deployment
- Configuring Oracle HTTP Server and OC4J for Global Deployment
- Configuring Oracle Application Server Portal for Global Deployment
- Configuring Oracle Application Server Wireless for Global Deployment
- Configuring Oracle Application Server Single Sign-On for Global Deployment
- Configuring Oracle Application Server Forms Services for Global Deployment
- Configuring Oracle Application Server Reports Services for Global Deployment
- Configuring Oracle Application Server Discoverer for Global Deployment
- Configuring Oracle Business Components for Java for Global Deployment
- Configuring a Centralized Unicode-enabled Database to Support Global Deployment

Installing Oracle Application Server for Global Deployment

In addition to the schemas of the infrastructure components, such as Oracle Internet Directory (OID) and Distributed Configuration Management (DCM), the Oracle Application Server Infrastructure database stores data pertaining to many Oracle Application Server middle-tier components that are installed on top of it. These components include Portal, Wireless, Forms, Reports, and Discoverer. Therefore, it is important to choose the correct database character set for the infrastructure

database at install time so that all of the dependent components are able to provide the same level of global support.

During the installation of the Oracle Application Server infrastructure database, you are prompted to choose the database character set you would like to use for the database. There are two basic scenarios that will determine which choice is best for your environment:

- If your environment is intended to support multiple languages in a single global instance of the Oracle Application Server infrastructure, choosing UTF-8 as the character set for the infrastructure database is highly recommended. Even if you only support a single language, such as English, you may choose UTF-8 as the database character set. The implications of choosing UTF-8 include, but are not limited to, the following:

  1. Databases with UTF-8 as the database character set are slightly slower than those with single-byte character sets. The performance impact is due to UTF-8 being a multibyte character set and the increase in the number of character set conversions between the middle tier and the database.

  2. Web pages served through `mod_plsql` must be encoded in UTF-8. However, there are some browsers, such as Netscape 4.7x and those for mobile devices, that may have some problems supporting UTF-8. Products that deliver Web pages through `mod_plsql` include Single Sign-On and Portal.

- If your environment is intended to support a single language or a group of languages that share the same native character set, you can choose the character set that is most commonly used for these languages as an alternative to UTF-8. For example, you can choose WE8MSWIN1252 if you are only interested in supporting Western European languages. You can choose JA16SJIS if you are interested in supporting Japanese and English.

During installation of any Oracle Application Server 10*g* installation type, support for user-selected languages is automatically installed and configured. It includes the translation files and fonts being used in the product. Additionally, command-line language installation is still supported.

If the required fonts are not available after installation, you can copy them from the Utilities CD included in the Oracle Application Server CD pack, or from `http://metalink.oracle.com` into the `$`*`ORACLE_HOME`*`/jdk/jre/lib/fonts` directory.

Configuring Oracle HTTP Server and OC4J for Global Deployment

This section contains the following topics related to configuring Oracle HTTP Server for multilingual support:

- About Manually Editing HTTP Server and OC4J Configuration Files
- Configuring the NLS_LANG Parameter
- Configuring Transfer Mode for mod_plsql Runtime
- Configuring the Runtime Default Locale

About Manually Editing HTTP Server and OC4J Configuration Files

If you edit Oracle HTTP Server or OC4J configuration files manually, instead of using Oracle Enterprise Manager, you must use the DCM command-line utility `dcmctl` to notify the DCM repository of the changes. Otherwise, your changes will not go into effect and will not be reflected in the Enterprise Manager consoles. The commands are as follows:

- To notify the DCM repository of changes made to Oracle HTTP Server configuration files:

  ORACLE_HOME/dcm/bin/dcmctl updateConfig ohs

- To notify the DCM repository of changes made to OC4J configuration files:

  ORACLE_HOME/dcm/bin/dcmctl updateConfig oc4j

- To notify the DCM repository of changes made to both Oracle HTTP Server and OC4J configuration files:

  ORACLE_HOME/dcm/bin/dcmctl updateConfig

Before you change configuration parameters, manually or using Oracle Enterprise Manager, you can save the current state of Oracle HTTP Server and OC4J configuration files and installed J2EE applications with the following command:

ORACLE_HOME/dcm/bin/dcmctl saveInstance -dir *directory_name*

You can then restore the state and back out of any subsequent changes that were made using the following command:

ORACLE_HOME/dcm/bin/dcmctl restoreInstance -dir *directory_name*

Configuring the NLS_LANG Parameter

The NLS_LANG parameter controls the language, territory, and character set used for database connections in an Internet application. Specify the value of NLS_LANG in the following format, including the punctuation as shown:

```
language_territory.characterset
```

language, *territory*, and *characterset* must be valid Oracle language, territory, and character set names. The specified language and territory are used to initialize the locale that determines the default date and time formats, number formats, and sorting sequence in a database session. The Oracle9*i* database converts data to and from the specified character set when it is retrieved from or inserted into the database.

> **See Also:** *Oracle9i Globalization Support Guide* in the Oracle Database Documentation Library for a list of valid Oracle language, territory, and character set names

You can specify the NLS_LANG parameter in the Oracle HTTP Server and OC4J files. The Oracle HTTP Server and OC4J files where NLS_LANG can be specified are as follows:

- $ORACLE_HOME/Apache/Apache/conf/httpd.conf

  This is the configuration of Oracle HTTP Server powered by Apache, and it defines the environment variables that are passed to Apache modules. If you want to explicitly specify the NLS_LANG parameter for CGI scripts such as Perl and SSI pages, you can add the following line to this file:

  ```
  SetEnv NLS_LANG language_territory.characterset
  ```

  Oracle HTTP Server is already configured to use the NLS_LANG shell environment variable in CGI scripts and SSI pages when NLS_LANG is not explicitly specified as described above. It does so by putting the following line into this file:

  ```
  PassEnv NLS_LANG
  ```

- $ORACLE_HOME/Apache/Apache/bin/apachectl

  This is the Oracle HTTP Server startup script used in UNIX. If you want to start Oracle HTTP Server directly from apachectl, you can specify the following line in this script file to define an NLS_LANG value:

  ```
  NLS_LANG=language_territory.characterset; export NLS_LANG
  ```

- $ORACLE_HOME/opmn/conf/opmn.xml

  Oracle Process Management and Notification (OPMN) is used to manage Oracle HTTP Server and OC4J instances. The opmn.xml configuration file allows you to specify the NLS_LANG environment variable for Oracle HTTP Server and OC4J processes through the following XML construct:

```
<environment>
...
<prop name="NLS_LANG" value="language_territory_characterset" />
...
</environment>
```

  This construct can be specified at the Oracle Application Server instance level where it applies to all Oracle HTTP Server and OC4J instances belonging to the Oracle Application Server instance. It can also be specified for the individual Oracle HTTP Server or OC4J instance where it only applies to the corresponding instance.

- $ORACLE_HOME/Apache/modplsql/conf/dads.conf

  This file defines database access descriptors (DADs) for mod_plsql to use when creating a database connection. You can specify the NLS_LANG value for the corresponding DAD. For example, you can specify the NLS_LANG value for the /pls/scott DAD as follows:

```
<Location /pls/scott>
     SetHandler pls_handler
     Order deny, allow
     Allow from all
     PlsqlDatabasePassword          tiger
     PlsqlDatabaseUsername          scott
     PlsqlDocumentPath              docs
     PlsqlEnableConnectionPooling   On
     PlsqlNlsLanguage             <NLS_LANG value>
</Location>
```

  Note that, when the Transfer Mod of a DAD is CHAR instead of RAW, the NLS_LANG character set of the DAD should be the same as that of the database character set for mod_plsql to work properly.

- $ORACLE_HOME/Apache/Jserv/etc/jserv.properties

  If JServ is needed in your environment, then you need to add or modify the following line in this file to define the appropriate NLS_LANG value:

```
wrapper.env=NLS_LANG=language_territory.characterset
```

If you do not explicitly specify the NLS_LANG environment variable in these files as described above, Oracle HTTP Server and OC4J will use the value set as follows:

- On UNIX: The NLS_LANG shell environment variable when Oracle HTTP Server and OC4J are invoked
- On Windows: The NLS_LANG registry key at \\HKEY_LOCAL_MACHINE\SOFTWARE\ORACE\HOME*n* in the Win32 registry

Preconfigured NLS_LANG Values

The Oracle Application Server installation pre-configures NLS_LANG values in the following Oracle HTTP Server and OC4J files for you based on the locale of the runtime environment on which the product is installed.

- $ORACLE_HOME/Apache/Apache/bin/apachectl (for UNIX platforms)
- $ORACLE_HOME/opmn/conf/opmn.xml/opmnctl (for UNIX platforms)

The pre-configured NLS_LANG values in the apachectl and opmnctl scripts are specified as follows:

```
NLS_LANG=${NLS_LANG=language_territory.characterset}; export NLS_LANG
```

The above line means that the pre-configured NLS_LANG values are used only when the shell environments from which the scripts are invoked have no defined the NLS_LANG environment variable. If you want to use an NLS_LANG value regardless of the shell environment, you can change this line to:

```
NLS_LANG=language_territory.characterset; export NLS_LANG
```

The NLS_LANG parameter controls the locale of the runtime environment on which OPMN runs. It should correspond to the default locale of the middle-tier runtime environment, which is the default locale of the operating system. The same NLS_LANG parameter is inherited by the OPMN managed processes, such as Oracle HTTP Server and OC4J, unless it is explicitly specified with a different value in opmn.xml.

For Windows platforms, the pre-configured NLS_LANG is automatically registered in the Win32 registry as the NLS_LANG registry key at \\HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\HOME*n*. The NLS_LANG value in this registry key controls the locale of the runtime environment on which OPMN and its managed processes run.

The pre-configured NLS_LANG values are the best values derived from the runtime locale during product installation, and may not represent the appropriate value for your Oracle HTTP Server and OC4J configurations. You may need to alter these values according to your specific requirements and runtime environments.

Setting NLS_LANG for a Monolingual Application Architecture

Set the NLS_LANG parameter to specify the language, territory, and character set that correspond to the locale that its middle-tier server is configured to serve. If most clients are running on Windows platforms, then it is a good practice to use the NLS_LANG character set that corresponds to the Windows code page of the locale. For example, when you configure the middle tier server to serve Japanese clients, then specify the following value for NLS_LANG:

```
JAPANESE_JAPAN.JA16SJIS
```

JA16SJIS corresponds to code page 932 of the Japanese Windows operation system.

Table 5–1 lists the NLS_LANG values for the most commonly used locales.

Table 5–1 NLS_LANG Values for Commonly Used Locales

Locale	NLS_LANG Value
Arabic (Egypt)	ARABIC_EGYPT.AR8MSWIN1256
Arabic (U.A.E.)	ARABIC_UNITED ARAB EMIRATES.AR8MSWIN1256
Chinese (Taiwan)	TRADITIONAL CHINESE_TAIWAN.ZHT16MSWIN950
Chinese (P.R.C.)	SIMPLIFIED CHINESE_CHINA.ZHS16GBK
Czech	CZECH_CZECH REPUBLIC.EE8MSWIN1250
Danish	DANISH_DENMARK.WE8MSWIN1252
Dutch	DUTCH_THE NETHERLANDS.WE8MSWIN1252
English (United Kingdom)	ENGLISH_UNITED KINGDOM.WE8MSWIN1252
English (U.S.A.)	AMERICAN_AMERICA.WE8MSWIN1252
Finnish	FINNISH_FINLAND.WE8MSWIN1252
French (Canada)	CANADIAN FRENCH_CANADA.WE8MSWIN1252
French (France)	FRENCH_FRANCE.WE8MSWIN1252
Germany (German)	GERMANY_GERMAN.WE8MSWIN1252
Greek	GREEK_GREECE.EL8MSWIN1253

Table 5–1 NLS_LANG Values for Commonly Used Locales (Cont.)

Locale	NLS_LANG Value
Hebrew	HEBREW_ISRAEL.IW8MSWIN1255
Hungarian	HUNGARIAN_HUNGARY.EE8MSWIN1250
Italian (Italy)	ITALIAN_ITALY.WE8MSWIN1252
Japanese	JAPANESE_JAPAN.JA16SJIS
Korean	KOREAN_KOREA.KO16MSWIN949
Norwegian	NORWEGIAN_NORWAY.WE8MSWIN1252
Polish	POLISH_POLAND.EE8MSWIN1250
Portuguese (Brazil)	BRAZILIAN PORTUGUESE_BRAZIL.WE8MSWIN1252
Portuguese (Portugal)	PORTUGUESE_PORTUGAL.WE8MSWIN1252
Romanian	ROMANIAN_ROMANIA.EE8MSWIN1250
Russian	RUSSIAN_CIS.CL8MSWIN1251
Slovak	SLOVAK_SLOVAKIA.EE8MSWIN1250
Spanish (Spain)	SPANISH_SPAIN.WE8MSWIN1252
Spanish (Latin American)	LATIN AMERICAN SPANISH_AMERICA.WE8MSWIN1252
Swedish	SWEDISH_SWEDEN.WE8MSWIN1252
Thai	THAI_THAILAND.TH8TISASCII
Turkish	TURKISH_TURKEY.TR8MSWIN1254

Setting NLS_LANG for a Multilingual Application Architecture

The language and territory components of the `NLS_LANG` parameter are not as important in the multilingual application architecture as they are in the monolingual application architecture. A multilingual application needs to handle different locales dynamically and cannot rely on fixed settings. The application should always use the UTF-8 character set so that Unicode data can be retrieved from and inserted into the database. An example of an appropriate value for `NLS_LANG` in a multilingual deployment is:

```
NLS_LANG=AMERICAN_AMERICA.UTF8
```

Configuring Transfer Mode for mod_plsql Runtime

The transfer mode of each database access descriptor (DAD) of the mod_plsql runtime enables PL/SQL to construct HTML content and process HTML form input in different character sets. You must set the transfer mode with the appropriate value.

It is important to configure the transfer mode for the mod_plsql module in the $ORACLE_HOME/Apache/modplsql/cfg/dads.conf file where the DADs are specified.

The mod_plsql module supports two transfer modes that you can configure in a DAD:

- CHAR mode: This is a default mode where dynamic HTML content is sent as VARCHAR2 data from the database to mod_plsql. In this mode, the NLS_LANG character set must be the same as that of the back-end database character set.
- RAW mode: Dynamic HTML content is sent as RAW data from the database to mod_plsql and is subject to character set conversion in the database server where the PL/SQL procedures and PSPs run. Character set conversion happens only when the HTML page encoding is specified, either by the NLS_LANG character set or by the charset parameter specified in the OWA_UTIL.MIME_HEADER() function call.

You should turn on the RAW transfer mode in a DAD for both monolingual and multilingual Internet applications as follows:

```
<Location /pls/scott>
        SetHandler pls_handler
        Order deny,allow
        Allow from all
            PlsqlDatabasePassword          tiger
            PlsqlDatabaseUsername          scott
            PlsqlDatabaseConnectString     local
            PlsqlDocumentPath              docs
            PlsqlEnableConnectionPooling   On
            PlsqlNlsLanguage               AMERICAN_AMERICA.UTF8
            PlsqlTransferMode              RAW
</Location>
```

Note that if the value of PlsqlNlsLanguage has a space in it, the value must be enclosed in quotation marks. For example:

```
PlsqlNlsLanguage "SIMPLIFIED CHINESE_CHINA.ZHS16GBK"
```

Configuring the Runtime Default Locale

This section describes how to initialize the runtime default locale for runtime environments that Oracle Application Server supports:

- mod_jserv Runtime for Java
- OC4J Java Runtime
- mod_plsql Runtime for PL/SQL and PL/SQL Server Pages
- mod_perl Runtime for Perl Scripts
- C/C++ Runtime

The default locale of a runtime environment controls the default locale-sensitive behavior of the applications, such as the character set used in file I/O operations, the language of the user interface, and the date format used. It needs to be properly set in order for applications relying on the default runtime locale to run with the expected locale-sensitive behavior. The default runtime locale is usually inherited from the default locale of the operating system or the locale of the runtime process.

The default runtime locale should be used as the user's preferred locale for monolingual applications. For multilingual applications, the default runtime locale is used for any server-side I/O operations, such as logging messages.

mod_jserv Runtime for Java

For UNIX platforms, the `LANG` or `LC_ALL` variable defines the following:

- The POSIX (also known as XPG4) locale used for a process
- How Java VM initializes its default locale

To configure the Java VM for JServ, define the `LANG` or `LC_ALL` environment variable with a POSIX locale name in the `jserv.properties` file. For example, the following line in `jserv.properties` defines Japanese (Japan) to be the default locale of Java VM for JServ on UNIX:

```
wrapper.env=LANG=ja_JP
```

The values for the `LANG` and `LC_ALL` environment variables should refer to the same POSIX locale available in your operating system. The `LC_ALL` environment variable always overrides the `LANG` environment variable if they are different.

The regional settings of the Control Panel control the default locale of the Java VM for JServ on Windows platforms. Change the regional settings to the desired locale from the Control Panel before starting Oracle HTTP Server.

OC4J Java Runtime

Define the LANG or LC_ALL environment variable with a POSIX locale name in $ORACLE_HOME/opmn/conf/opmn.xml. For example, the following line within the <environment> tags in opmn.xml defines Japanese (Japan) to be the default locale of Java VM for OC4J on Solaris:

```
<environment>
...
<prop name="LANG" value="ja_JP" />
...
</environment>
```

The regional settings of the Control Panel control the default locale of the Java VM for OC4J on Windows platforms. Change the regional settings to the desired locale from the Control Panel before starting Oracle HTTP Server.

mod_plsql Runtime for PL/SQL and PL/SQL Server Pages

PL/SQL and PL/SQL Server Pages run on an Oracle9*i* database in the context of a database session. Therefore, the NLS_LANG parameter controls the runtime default locale. The NLS_LANG parameter should be configured as described in "Configuring the NLS_LANG Parameter".

mod_perl Runtime for Perl Scripts

Perl scripts run on the Perl interpreter that the mod_perl module provides. The locale support in Perl is based on the POSIX locale available in the operating system. It uses the underlying POSIX C libraries as a foundation. To configure the Perl runtime default locale, follow the procedure described for the C/C++ runtime.

See Also:

- "C/C++ Runtime"
- *Oracle HTTP Server Administrator's Guide* for more information about how Perl scripts use POSIX locales

C/C++ Runtime

The C/C++ runtime uses the POSIX locale system that the operating system provides. You can configure the locale system by defining the LC_ALL or LANG environment variable. Define LC_ALL with a valid locale value that the operating system provides. These values are different on different operating systems.

See Also: Table 5–1 for a list of commonly used POSIX locales for Solaris

For UNIX platforms, define LC_ALL as follows:

- In $ORACLE_HOME/Apache/Apache/conf/httpd.conf, add the following line:

```
PassEnv LC_ALL
```

- In $ORACLE_HOME/Apache/Apache/bin/apachectl, add the following line:

```
LC_ALL=${LC_ALL=OS_locale}; export LC_ALL
```

For Windows platforms, the POSIX locale should inherit its value from the regional settings of the Control Panel instead of being specified in the LC_ALL environment variable. Change the regional settings to change the default runtime POSIX locale.

Configuring Oracle Application Server Portal for Global Deployment

Oracle Application Server Portal is designed to allow application development and deployment in different languages. OracleAS Portal is configured with the languages that are selected in the Oracle Universal Installer (OUI) during the Oracle Application Server middle-tier installation. Languages that are configured show up in the **Set Language** portlet.

To configure additional languages after installation, the OracleAS Portal Configuration Assistant (OPCA) must be used in LANGUAGE mode. Once you have installed a language, OracleAS Portal allows you to specify the preferred locale and territory to be used for that language; for example, Australian English or Canadian French. See Appendix A for a list of languages and abbreviations that are available for OracleAS Portal.

See Also:

- *Oracle Application Server Portal Configuration Guide* for details and usage information for OPCA
- *Oracle Application Server 10g mod_plsql User's Guide*

Configuring Oracle Application Server Wireless for Global Deployment

When users access wireless services from their mobile devices, Oracle Application Server Wireless uses the user profile information from Oracle Internet Directory (OID) to determine the user's preferred language. Administrators can select the language when creating a new user through the Oracle Application Server Wireless Tools. Users can change their preferred language through the Wireless Customization Tool.

Configuring Encoding for Outgoing Email Messages

When users send email messages from their mobile devices, Oracle Application Server Wireless sends the messages in the encoding specified in the encoding parameter of the PIM / Mail service.

You can change the default encoding for outgoing email messages by modifying the `ORACLE_SERVICES_PIM_MAIL_MESSAGE_ENCODING` parameter of the PIM / Mail master service.

Configuring Oracle Application Server Single Sign-On for Global Deployment

Oracle Application Server Single Sign-On is automatically configured with the languages that are selected in the Oracle Universal Installer (OUI) during the Oracle Application Server Infrastructure installation. To configure additional languages after installation, you need to execute the following command:

ORACLE_HOME`/jdk/bin/java -jar` *ORACLE_HOME*`/sso/lib/ossoca.jar langinst` *lang make_lang_avail ORACLE_HOME*

In this command, *lang* specifies the abbreviation code for the language to be installed. See Appendix A for a list of languages and their corresponding abbreviations. The value of *make_lang_avail* specifies whether or not to make the language available. Enter `1` to make the language available, `0` otherwise.

> **See Also:** *Oracle Application Server Single Sign-On Administrator's Guide* for detailed steps on configuring additional languages

Configuring Oracle Application Server Forms Services for Global Deployment

The NLS_LANG parameter controls the language, territory, and character set that an Internet application uses for database connections. Specify the value of NLS_LANG in the following format, including the punctuation as shown:

```
language_territory.characterset
```

language, *territory*, and *characterset* must be valid Oracle language, territory, and character set names. The specified language and territory are used to initialize the locale that determines the default date and time formats, number formats, and sorting sequence in a database session. Oracle Net converts data to and from the specified character set when it retrieves data from or inserts data into the database.

You can set the NLS_LANG parameter in the $ORACLE_HOME/forms90/server/default.env file. If you do not set the NLS_LANG parameter in the default.env file, then Forms Services uses the value set as follows:

- On UNIX: The NLS_LANG shell environment variable when Forms Server is invoked
- On Windows: The NLS_LANG setting at the \\HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE*FormsServerOracle_HOME* in the Win32 registry

You can have different NLS_LANG settings on the same Forms Server by specifying an alternate environment file. Use the envFile parameter in the formsweb.cfg file. To do this:

1. Create two environment configuration files under $ORACLE_HOME/forms90/server. For example, an American environment configuration file (en.env) should contain the following lines:

   ```
   NLS_LANG=AMERICAN_AMERICA.US7ASCII
   FORMS90_PATH=d:\us
   ```

   A Japanese environment configuration file (ja.env) should contain the following lines:

   ```
   NLS_LANG=JAPANESE_JAPAN.JA16SJIS
   FORMS90_PATH=d:\ja
   ```

2. In the `formsweb.cfg` file (`$ORACLE_HOME/forms90/server`), set the `envFile` parameter for the alternative setting. For example:

```
[ja]
envFile=ja.env

[en]
envFile=en.env
```

Then specify the configuration name in the URL for your forms servlet as follows:

```
http://formsservermachine/forms90/f90servlet?config=ja
http://formsservermaching/forms90/f90servlet?config=en
```

See Also: *Oracle Application Server Forms Services Deployment Guide*

Configuring Oracle Application Server Reports Services for Global Deployment

The `NLS_LANG` parameter controls the language, territory, and character set used for database connections in an Oracle Application Server Reports Services application. Specify the value of `NLS_LANG` in the following format, including the punctuation as shown:

```
language_territory.characterset
```

`language`, *`territory`*, and *`characterset`* must be valid Oracle language, territory, and character set names. The specified language and territory are used to initialize the locale that determines the default date and time formats, number formats, and sorting sequence in a database session. Oracle Net converts data to and from the specified character set when it retrieves data from or inserts data into the database.

Oracle Application Server Reports Services uses the value of the `NLS_LANG` parameter set as follows:

- On UNIX: The `NLS_LANG` shell environment variable when Oracle Application Server Reports Services is invoked. The default `NLS_LANG` value is set in *`$ORACLE_HOME`*`/bin/reports.sh`.

- On Windows: The value of NLS_LANG set at \\HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\%*ReportsORACLE_HOME*% in the Win32 registry
- For dynamic environment switching: In the Reports Server configuration file through the environment element. On UNIX, the default NLS_LANG value in the reports.sh file needs to be commented out to enable this feature.

See Also:

- *Oracle Application Server Reports Services Publishing Reports to the Web* for more information about dynamic environment switching
- *Oracle9i Globalization Support Guide* in the Oracle Database Documentation Library for more information about these parameters

Configuring Oracle Application Server Discoverer for Global Deployment

Oracle Application Server Discoverer can simultaneously support users with different locales. Users may explicitly control the locale used for the user interface, or they may allow Oracle Application Server Discoverer to automatically determine a default. The order of precedence for determining the language and locale is:

1. Language and locale settings specified in the URL for Oracle Application Server Discoverer
2. Language and locale settings specified in the Discoverer Connection (this is part of the Oracle Application Server Discoverer integration with Oracle Application Server Single Sign-On)
3. Language and locale setting specified in the user's browser
4. Language and locale of Oracle Application Server

See Also: *Oracle Application Server Discoverer Configuration Guide* for more information on using URL parameters with Discoverer.

Configuring Oracle Business Components for Java for Global Deployment

You can set the following Oracle Business Components for Java (BC4J) properties:

- jbo.default.language
- jbo.default.country

Their default values are `en` and `US`, respectively.

You can set them at the command line, by modifying the `jboserver.properties` file, or with an applet parameter tag. All sessions share this locale to display messages.

Configuring a Centralized Unicode-enabled Database to Support Global Deployment

You can set up the centralized Oracle9*i* database to store Unicode data in the following ways:

- As UTF-8 in the SQL `CHAR` data types (`CHAR`, `VARCHAR2`, and `CLOB`)
- As UTF-16 in the SQL `NCHAR` data types (`NCHAR`, `NVARCHAR2`, and `NCLOB`)

See Also:

- *Oracle9i Globalization Support Guide* in the Oracle Database Documentation Library
- *Oracle9i SQL Reference* in the Oracle Database Documentation Library

It is good practice to configure the centralized Oracle9*i* database to support the following:

- UTF-8 in the SQL `CHAR` data types

  Specify AL32UTF8 for the database character set when you create the centralized database.

- UTF-16 in the SQL `NCHAR` data types

  Specify AL16UTF16 for the national character set when you create the centralized database.

Example 5–1 shows part of a CREATE DATABASE statement that sets the recommended database character set and national character set.

Example 5–1 Specifying the Database Character Set and the National Character Set

```
CREATE DATABASE myunicodedatabase
    CONTROL FILE REUSE
    LOGFILE '/u01/oracle/ubfdb/redo01.log' SIZE 1M REUSE
    '/u01/oracle/utfdb/redo02.log' SIZE 1M REUSE
    DATAFILE '/u01/oracle/utfdbsystem01.dbf' SIZE 10M REUSE
    AUTOEXTENT ON
```

```
NEXT 10M MAXSIZE 200M
CHARACTER SET UTF8
NATIONAL CHARACTER SET AL16UTF16
... ;
```

6

A Multilingual Demo for Oracle Application Server

This chapter describes the World-of-Books demo that is provided with Oracle Application Server.

This chapter contains the following topics:

- Description of the World-of-Books Demo
- Architecture and Design of the World-of-Books Demo
- Building, Deploying, and Running the World-of-Books Demo
- Locale Awareness of the World-of-Books Demo
- Encoding HTML Pages for the World-of-Books Demo
- Handling HTML Form Input for the World-of-Books Demo
- Encoding URLs in the World-of-Books Demo
- Formatting HTML Pages in the World-of-Books Demo
- Accessing the Database in the World-of-Books Demo
- Organizing the Content of HTML Pages in the World-of-Books Demo

Description of the World-of-Books Demo

The World-of-Books (WOB) demo demonstrates how to write a multilingual Web application and deploy it on the Oracle Application Server J2EE container. The application consists of the following Web sites:

- An online store that sells books in different languages

- An online Chinese book supplier administration site that represents book Supplier A
- An online global book supplier administration site that represents book Supplier B

The online bookstore is a multilingual Web application that interacts with customers. It allows customers to view books, check prices, and place orders. The application uses HTTP connections to send orders as XML documents to the suppliers. The online book supplier administration sites are Web applications that the book suppliers use to get orders from the bookstore, to send order status reports to the bookstore, and to notify the bookstore about newly available books.

The online bookstore supports 60 locales. Customers in these locales can use the online bookstore with their preferred language and cultural conventions. The online book supplier administration sites are in English only.

Architecture and Design of the World-of-Books Demo

The WOB demo serves customers with different locale preferences. It is mainly written in Java, using Java Servlets, Java beans, and Java Server Pages (JSPs). It uses the Unicode capabilities available in Java, XML, JDBC, and the Oracle9*i* database to support multilingual data and a multilingual user interface.

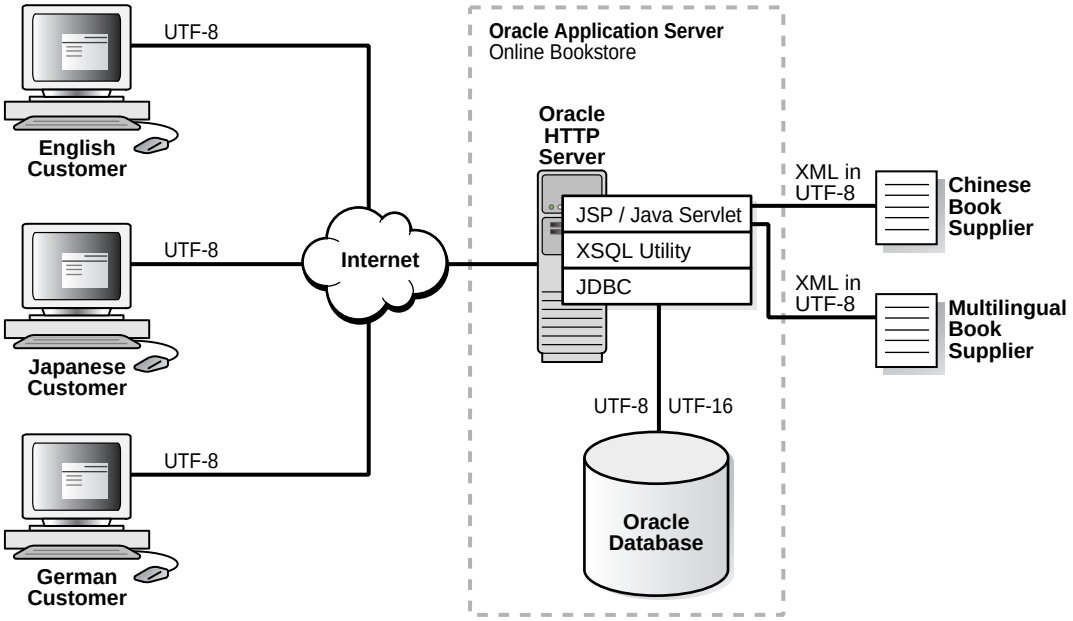
This section contains the following topics:

- World-of-Books Architecture
- World-of-Books Design
- World-of-Books Schema Design

World-of-Books Architecture

Figure 6–1 shows the architecture of the WOB demo.

Figure 6–1 World-of-Books Architecture



The application architecture can be summarized as follows:

- JSPs generate dynamic content in UTF-8 encoded HTML pages.
- Java Servlets and Java Beans implement the business logic.
- The Oracle9*i* database stores book and customer information.
  - Oracle Text enables locale-sensitive, full-text searches on the contents of books.
  - The SQL `NVARCHAR2` datatype stores multilingual book information.
- The Oracle JDBC driver (either OCI or the Thin driver) accesses Unicode data stored in the Oracle9*i* database. The data can be encoded in UTF-8 if the target column is of a SQL `CHAR` datatype, or the data can be encoded in UTF-16 if the target column is of a SQL `NCHAR` datatype.
- The document format for communications between the online bookstore and the book suppliers is UTF-8 encoded XML.

Figure 6–1 shows the WOB application on Oracle Application Server. The processing character set for the WOB application is UTF-16. The application uses XML messages to communicate with the Chinese book supplier and the multilingual book supplier. The XML messages are encoded in the UTF-8 character set. English, Japanese, and German customers connect to the WOB application through the Internet. The application serves all of the customers HTML pages encoded in the UTF-8 character set.

World-of-Books Design

Table 6–1 shows the Java programs that contain most of the internationalization features for the WOB application. The programs are located in `$WOB_HOME/src/oracle/demo/wob2/wob`.

Table 6–1 Java Programs that Contain Internationalization Features for the World-of-Books Application

Java Program	Purpose
`beans/Localizer.java`	Contains all locale-related information and locale-sensitive methods for a specific user session
`LocaleUtil.java`	Contains methods for retrieving static information such as the list of supported languages and the list of supported countries
`Language.java`	Contains information about a language and its properties such as the writing direction (right to left or left to right)
`Country.java`	Contains information about a country and its properties such as currency and date formats

The `Language` and `Country` classes are used only by the `LocaleUtil` class to represent a language and a country. The `Localizer` bean calls the `LocaleUtil` class for all static information about locale and character set, such as the default language and the default encoding used for the user interface of the bookstore. The `LocaleUtil` class reads the properties resource bundle, `wob.properties`, to initialize all static information for the online bookstore. The `wob.properties` file is located in `$WOB_HOME`.

Most of the JSPs for the online bookstore include the `header.jsp` file, which uses the `Localizer` Java bean to keep locale information for a session. JSPs call the `Localizer` Java bean to perform all locale-sensitive operations such as formatting a date, encoding a URL, and converting HTML form parameters to Java strings. Some JSPs also call the `LocaleUtil` static class to get information such as the list of available languages and the list of currencies used for a specific country.

World-of-Books Schema Design

The database schema for the WOB demo consists of the following tables:

- `customers`: Stores the user profile for each WOB user
- `books`: Stores the information about each book
- `docs`: Stores the content of each book so that customers can search the content of the books

Table 6–2 describes the `customers` table. When a registered user is logged in, the online bookstore uses the locale preferences in the customer table in the `Localizer` bean.

Table 6–2 Description of the customers Table

Column	Datatype	Description
`custid`	`VARCHAR2(50)`	User's name (this is the primary key\)
`locale`	`VARCHAR2(10)`	User's preferred locale, in ISO locale format (for example, `en-US`)
`currency1`	`VARCHAR2(10)`	ISO locale whose default primary currency is used by the user
`currency2`	`VARCHAR2(10)`	ISO locale whose default dual currency is used by the user
`timezone`	`VARCHAR2(50)`	User's time zone (for example, Asia/Hong Kong)
`encoding`	`VARCHAR(40)`	User's HTML page encoding (for example, UTF-8)

Table 6–3 describes the `books` table. The `NVARCHAR2` datatype is used for the title, author, short description, and publisher of the book. By storing this information as Unicode in the `NVARCHAR2` datatype, the WOB demo can support books in languages from around the world. The `nsort` column is used for queries about books so that the list is returned in an order appropriate for the locale.

Table 6–3 Description of the books Table

Column	Datatype	Description
`langid`	`NUMBER(3)`	Language of the book
`bookid`	`NUMBER (10)`	Unique identifier of the book (this is the primary key)
`nsort`	`VARCHAR2(30)`	Locale-sensitive sorting sequence used in the `NLSSORT()` SQL function for the book

Table 6–3 Description of the books Table (Cont.)

Column	Datatype	Description
title	NVARCHAR(300)	Book title
author	NVARCHAR(300)	Book author
descpt	NVARCHAR(2000)	Short description of the book
publisher	NVARCHAR(200)	Name of the book's publisher

Table 6–4 describes the docs table. It stores the contents of the books.

Table 6–4 Description of the docs Table

Column	Datatype	Description
bookid	NUMBER(10)	Unique identifier of the book (this is the primary key)
langid	NUMBER(3)	Language of the book
mimetype	VARCHAR2(50)	MIME type of the book
language	VARCHAR2(30)	Language of the contents of the book, using the Oracle NLS language naming convention
format	VARCHAR2(10)	Format of the contents of the book (TEXT or BINARY)
cset	VARCHAR2(30)	Character set of the contents of the book
doc	BLOB	Contents of the book

Indexes have been built for these tables. The following SQL files are used to create these tables and build the corresponding indexes. They are located in the $WOB_HOME/schema directory:

- customers.sql
- books.sql
- docs.sql

Oracle Text requires the language, format, cset, and doc columns of the docs table to build a full-text search index on the docs table. The ctxidx.sql and ctxsys.sql scripts are used to set up the full-text search index. They are located in $WOB_HOME/schema/ctx.

See Also: *Oracle9i Globalization Support Guide* in the Oracle Database Documentation Library for more information about building a full-text search index

Installing the World-of-Books Demo

The World-of-Books demo is available as a zip file that you can download from the Oracle Technology Network (OTN) at `http://otn.oracle.com/tech/java/oc4j/demos/`. After you download the `globalization_wob_demo.zip` file, unzip the file as follows:

- Go to the `$ORACLE_HOME/j2ee/home/demo` directory, or create it if it does not already exist.
- Copy the file to the `$ORACLE_HOME/j2ee/home/demo` directory.
- Unzip the file.

Note: Environment variable references, such as `$ORACLE_HOME`, are shown in UNIX format. For Windows environments, use the `%ORACLE_HOME%` notation.

After unzipping the downloaded file, you should see the directory `globalization` under `$ORACLE_HOME/j2ee/home/demo`. This directory, `$ORACLE_HOME/j2ee/home/demo/globalization`, is the root directory of the World-of-Books demo. This root directory is referred to as `$WOB_HOME` throughout this chapter.

Building, Deploying, and Running the World-of-Books Demo

The source code and the build files of the World-of-Books demo are in the WOB demo home directory located in `$WOB_HOME`. Table 6–5 shows the directory structure under `$WOB_HOME`.

Note: Environment variable references, such as `$ORACLE_HOME`, are shown in UNIX format. For Windows environments, use the `%ORACLE_HOME%` notation.

Table 6–5 World-of-Books Directory Structure

Directory/Files	Description
`docroot`	Contains all static files such as HTML files, JSPs, and images
`docroot/wob`	Contains static files for the online bookstore Web application
`docroot/suppa`	Contains static files for the Chinese book supplier administration application
`docroot/suppb`	Contains static files for the global book supplier administration application
`src/oracle/demo/wob2`	Contains all Java programs
`src/oracle/demo/wob2/wob`	Contains Java programs for the online bookstore application
`src/oracle/demo/wob2/supp`	Contains Java programs shared by the two online supplier applications
`build.xml`	Builds the WOB demo
`README.TXT`	Contains useful information for building and deploying the WOB demo
`schema`	Contains SQL files to create and populate the database schema that the WOB demo uses
`j2ee_config`	Contains J2EE deployment files for the WOB demo
`etc`	Contains the configuration files for the WOB demo applications

This section contains the following topics:

- How to Build the World-of-Books Demo
- How to Deploy the World-of-Books Demo
- How to Run the World-of-Books Demo

How to Build the World-of-Books Demo

> **Note:** Commands for setting environment variables are based on the C shell convention. For Windows platforms, use the SET command at the DOS prompt to set environment variables. Additionally, replace forward slashes with backslashes in all directory paths.

To build the WOB demo:

1. Go to the $ORACLE_HOME/j2ee/home/demo/globalization directory.
2. Update the suppa.properties, suppb.properties, and wob.properties files in the $WOB_HOME/etc directory.
   - Replace <J2EE_HOME> with the full path where OC4J is installed. It should be $ORACLE_HOME/j2ee/home.
   - Replace <HOSTNAME> with the host name of your machine.
   - Replace <PORT> with the port number of your default Web site. By default, this should be 7778.
3. Set up the JAVA build environment by defining the JAVA_HOME and CLASSPATH environment variables. Oracle Application Server bundles JDK under $ORACLE_HOME/jdk so that you can use it for your JAVA_HOME.

   You can also use your own JDK. For example:

   ```
   % setenv ORACLE_HOME yourOracleHome
   % setenv JAVA_HOME $ORACLE_HOME/jdk
   % setenv J2EE_HOME $ORACLE_HOME/j2ee
   % copy $ORACLE_HOME/rdbms/jlib/xsul2.jar to $ORACLE_HOME/j2ee/home/lib
   ```

   Make sure that $ORACLE_HOME/bin is in your path directory. For example:

   ```
   % setenv PATH $ORACLE_HOME/bin:$PATH
   ```

4. Ensure that an Oracle9*i* database is available to load the schema and data for the WOB demo by defining the TWO_TASK environment variable to point to your database. For example, if you can access the database from SQL*Plus with the connect string iasdb, you can define the TWO_TASK environment variable as follows:

   ```
   % setenv TWO_TASK iasdb
   ```

5. Build the WOB demo by entering the ANT command from the $WOB_HOME directory.

   ```
   % ant
   ```

   The build process performs the following tasks:

   - Compiles all Java programs
   - Packages all of the static files and Java classes into an EAR file and a WAR file, which are used for deployment
   - Creates the WOB schema and populates it with the seed data that is provided

6. If you enabled Oracle Text in your database, then you can set up full text searches on book content by building the full text search index.

   ```
   % ant setupctx
   ```

How to Deploy the World-of-Books Demo

To deploy the WOB demo on Oracle Application Server J2EE:

1. Update $WOB_HOME/j2ee_config/data-sources.xml, which is used for database connection.
   - Replace <HOSTNAME> with the host name of the Oracle9*i* database server.
   - Replace <PORT> with the port number of the Oracle9*i* database server.
   - Replace <ORACLE_SID> with the SID of the Oracle9*i* database server.
   - Cut and paste the contents of data-sources.xml into $J2EE_HOME/config/data-sources.xml.
2. Update the configuration file using DCM as follows:

   ```
   dcmctl updateConfig
   ```

3. Deploy the application $WOB_HOME/lib/glln.ear using Oracle Enterprise Manager (OEM). Alternatively, you can deploy the application using dcmctl as follows:

   ```
   dcmctl deployApplication -file $WOB_HOME/lib/glln.ear
                            -application glln -component home
   ```

How to Run the World-of-Books Demo

The online bookstore requires one of the following browsers:

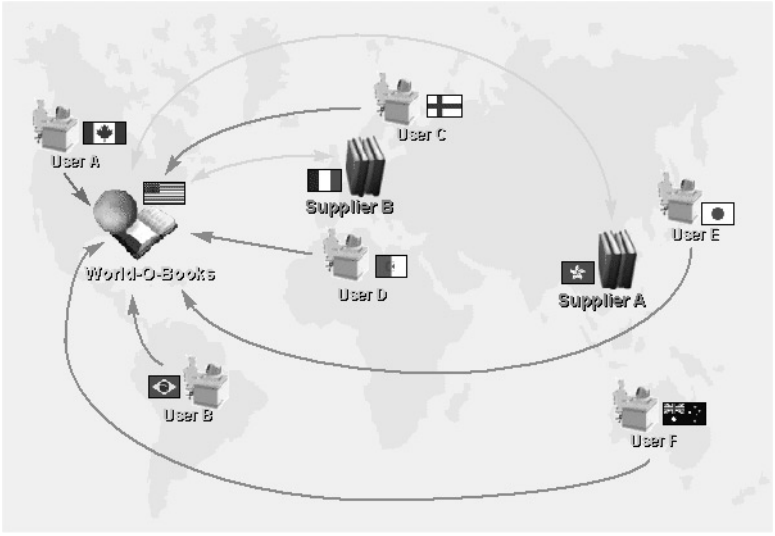
- Internet Explorer 5.0 or above
- Netscape 4.7 or above

The book supplier administration applications require Internet Explorer 5.0 or above.

To run the WOB demo, start the browser and enter the following URL:

`http://host_name:7778/g11n/imap.html`

You should see a screen similar to the following:

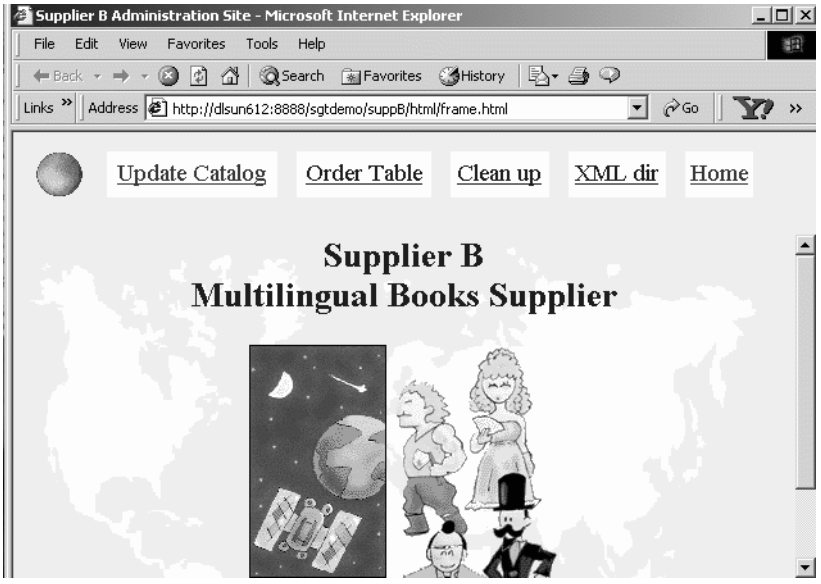


Select a link to start the desired application.

Image	Link Target
World-O-Books image	Online bookstore application
Supplier A image	Chinese book supplier administration application
Supplier B image	Global book supplier administration application

You can navigate the online bookstore as a registered customer or as a visitor.

If you click the Supplier B image, the following screen appears:



The links on the Supplier B administration site are as follows:

Link	Description
Update Catalog	Allows the supplier to send new book information to the online bookstore to update the bookstore catalog. It sends an XML file to the online bookstore.
Order Table	Allows the supplier to check for customer orders sent from the online bookstore and can update the order status.
Clean up	Restores the data to the initial state. All previous orders and newly added books are deleted.
XML dir	Lists the XML documents that have been sent to and from the online bookstore.
Home	Returns to the WOB home page.

Locale Awareness of the World-of-Books Demo

The World-of-Books online bookstore is fully aware of the user's locale. The application determines the user's locale and uses this locale to format dynamic HTML pages according to the user's language and cultural conventions.

This section contains the following topics:

- How World-of-Books Determines the User's Locale
- How World-of-Books Uses Locale Information in Localizer Methods
- How World-of-Books Sorts Query Results
- How World-of-Books Searches the Contents of Books

How World-of-Books Determines the User's Locale

The online bookstore determines the user's locale using three methods in the following order:

- If a customer has logged into the bookstore, it examines the locale associated with the customer's user profile and uses it as the preferred locale.
- Allows the user to enter the locale from the bookstore's user interface.
- Examines the Accept-Language HTTP header sent from the browser.

The `Localizer` bean has two properties, `AcceptLang` and `localeOverride`. The `AcceptLang` property indicates the Accept-Language header of the current HTTP request. The `localeOverride` property indicates whether a user has explicitly selected a locale, which is passed as a GET request parameter of the current HTTP request. The `header.jsp` file initializes the values of these properties as follows:

```
<jsp:useBean id="my" class="oracle.demo.wob2.wob.beans.Localizer"
scope="session" />
<jsp:setProperty name="my" property="AcceptLang"
value="<%=request.getHeader(\"Accept-Language\") %>" />
<jsp:setProperty name="my" property="localeOverride" value="<%=
request.getParameter(\"v_override\") %>" />
```

This initialization causes the `setAcceptLang()` and `setlocaleOverride()` methods of the `Localizer` bean to initialize the `Localizer` associated with the current HTTP request with the appropriate locale information. The application determines the current user's locale as follows, in this order:

1. If the user has already been logged in to the current HTTP session, it uses the locale preference in the user profile. The `isLoggedIn()` method of the `Localizer` determines whether the current user is logged in.
2. If the `localeOverride` property is not NULL, it uses the locale that this property indicates as the user locale.

3. If the AcceptLang property is not NULL, it uses the locale that this property indicates as the user locale.
4. Otherwise, it uses the default locale indicated by the default_language property from the wob.properties resource bundle. This default locale is initialized in LocaleUtil.java.

The displayFlags() method in the Localizer generates the HTML content that enables users to enter a locale by clicking one of the displayed flags. The header.jsp file calls this method.

How World-of-Books Uses Locale Information in Localizer Methods

After the Localizer is initialized with the user's locale, all methods of the Localizer are sensitive to the locale. Table 6–6 shows examples of locale-sensitive methods defined in the Localizer.

Table 6–6 Examples of Locale-Sensitive Methods of the Localizer Bean

Method	Example of Use
String formatDate()	The following JSPs use the formatDate() method: ■ welcome.jsp formats the system date that the welcome page displays. ■ History.jsp formats the date of the order history. ■ setting.jsp formats the date to be displayed when a registered user updates the user profile.
String getCurrency()	Changeprofile1.jsp gets the primary currency symbol to be displayed for the user profile modification screen.
String getDualCurrency()	Changeprofile1.jsp gets the alternate or dual currency symbol to be displayed for the user profile modification screen.
String getTimeZone()	myaccount.jsp displays the time zone of the current user.
String getDirection()	setting.jsp displays the direction that text is written, based on the current user.
String getMessage()	Most of the JSPs use this method to get the translated message that corresponds to the current locale from a resource bundle.
String getNLSLanguage()	search.jsp gets the Oracle language name used for the current locale and for submitting a language-sensitive search.

Other locale-sensitive functions are described in the following sections.

How World-of-Books Sorts Query Results

The order in which books are listed in the results of a query is sensitive to the current user's locale. The search template is as follows:

```
   SELECT  books.bookid,
           langmap.language,
           books.title,
           books.author,
           substr(books.descpt, 1, 50)
     FROM  books, langmap
    WHERE  <specific search criteria>
           books.langid = langmap.langid AND
           nlssort(books.title, 'NLS_SORT = '|| books.nsort) IS NOT NULL
 ORDER BY  langord(books.langid, 'Oracle_NLS_language'),
           nlssort(books.title, 'NLS_SORT='||books.nsort);
```

The `langmap` table maps language IDs to Oracle NLS language names and Oracle sort names used in the `NLSSORT` SQL function. The `$WOB_HOME/schema/langmap.sql` file creates the `langmap` table.

The `SELECT` statement orders the books with the `ORDER BY` clause as follows:

1. It groups the books by their languages, using the first sort key that the `langord` PL/SQL function returns. The `langord` function returns the smallest key value when the Oracle NLS language that corresponds to the current user's locale matches the language of the book. Thus the books are grouped so that the first group consists of books whose language corresponds to the user's locale.
2. Within each language group, it orders the books by the sort key that the `NLSSORT` SQL function returns. The `NLSSORT` function generates sort keys based on the linguistic order specified by the `NLS_SORT` parameter. The value of the `NLS_SORT` parameter is stored in the `nsort` column of the `books` table. Thus the books in the sorted group are ordered by the Oracle sort sequence name stored in the `nsort` column.

The application also orders lists in the user interface using locale information. For example, it uses the `displayLanguageOptions()` method of the `Localizer` bean to construct a list of languages so users can select a language. The `displayLanguageOptions()` method collates the languages in the list based on the locale-specific Java collator. This collator is constructed using the current locale represented by the `Localizer` bean. The following code gets the collation key of each language name in the current user's locale:

```
String[] languages = localeutil.getSupportedLanguagesArray();
    CollationKey[] keys = new CollationKey[languages.length];
```

```
for (int i = 0; i < languages.length; i++)
{
  keys[i] = collator.getCollationKey(getMessage(languages[i])
            + " [" + languages[i]);
}
```

After the keys array is filled with collation keys, the array is sorted based on the binary value of each key. The other methods that collate drop-down lists are `displayCountryOptions()`, `displayCurrencyOptions()`, and `displayScriptCountryVars()`.

How World-of-Books Searches the Contents of Books

The online bookstore allows users to search the contents of books in a locale-sensitive manner. The following query searches the contents of the books from the `docs` table:

```
SELECT  books.bookid,
        langmap.language,
        books.title,
        books.author,
        substr(books.descpt, 1, 50)
  FROM  books, langmap, docs
 WHERE  contains(docs.doc, 'search_key', 0) > 0 AND
        books.langid = langmap.langid AND
        nlssort(books.title, 'NLS_SORT = '|| books.nsort) IS NOT NULL

ORDER BY  langord(books.langid, 'Oracle_NLS_language'),
          nlssort(books.title, 'NLS_SORT='||books.nsort);
```

The `contains(docs.doc, 'search_key', 0)` function in the `WHERE` clause returns a positive value when the search key is found in the contents of a document stored in the `doc` column of the `docs` table. The rest of the query is similar to the query used for the book search.

Oracle Text by default uses the language of the search key as defined by the `NLS_LANGUAGE` session parameter. To conduct the search in a language-sensitive manner, `search.jsp` issues an `ALTER SESSION` statement to change the value of the `NLS_LANGUAGE` parameter to the value that the user specifies before submitting the content search query. The `ALTER SESSION` statement is as follows:

```
ALTER SESSION SET NLS_LANGUAGE=language;
```

Calling the `getParameter("v_language")` method of the `HTTPServletRequest` object obtains the language value, where `v_language` is a form input parameter from the advanced search screen.

Encoding HTML Pages for the World-of-Books Demo

In the online bookstore, an attribute of the `Localizer` bean stores the encoding used for HTML pages. The `default_encoding` property of the `wob.properties` resource bundle initializes the attribute with the default page encoding. By default, the online bookstore uses UTF-8 as the HTML page encoding to provide support for multilingual content.

To enforce UTF-8 as the page encoding in JSPs, define the appropriate Content-Type header. For the online bookstore, put the Content-Type page directive into `header.jsp` as follows:

```
<%@ page contentType="text/html;charset=UTF-8" %>
```

You only need to put this directive into `header.jsp` because all other JSPs that produce HTML output include `header.jsp`.

The online bookstore allows users to override the default encoding with the preferred encoding from the user profile. The user can choose a preferred page encoding from the user profile modification page. After the user logs in, the encoding attribute of the current `Localizer` bean is updated to the preferred encoding. To set the encoding in JSPs, `header.jsp` checks whether the user has logged in and calls the `HTTPServletResponse.setContentType()` method to overwrite the Content-Type header defined in the JSP page directive with the preferred encoding. The code is as follows:

```
<% if (my.isLoggedIn())
     response.setContentType("text/html; charset=" + my.getEncoding());
%>
```

The `getEncoding()` method of the `Localizer` bean (`my`) returns the preferred encoding from the current user's profile.

Handling HTML Form Input for the World-of-Books Demo

The online bookstore accepts multilingual text as HTML form input. The input can be a search key when the user wants to search for a book, or it can be a user name at login. The browser sends form input as a sequence of bytes in the same encoding as the HTML form. Converting the input to Java strings encoded as Unicode requires

the page encoding information. Because the page encoding is stored as an attribute of the `Localizer`, the conversion function is encapsulated in the Localizer class.

The `translateParameter()` method of the `Localizer` bean converts form input from the encoding indicated in the encoding attribute of the bean to a Java string. The method is as follows:

```
public String translateParameter(parameter_string)
{
    try {
      byte[] paramBytes = param.getBytes("ISO-8859-1");
      return new String(paramBytes, getEncoding());
    } catch (UnsupportedEncodingException e)
    {
      // return the same string if exception
    }
    return param;
}
```

The `getEncoding()` method of the `Localizer` bean returns the page encoding for HTML forms.

The JSPs call the `translateParameter()` method where form input is processed. For example, the following files call this method:

- `search.jsp` uses it to get a search key
- `updateprofile.jsp` uses it to get new user profile information
- `login.jsp` uses it to get the user name

Encoding URLs in the World-of-Books Demo

All URLs that are embedded in an HTML page must be encoded. They must use the same encoding as the HTML page. The `Localizer` bean is the best place to encapsulate the `encodeURL()` method. This method encodes a URL according to the encoding attribute of the `Localizer`.

The following JSPs call the `encodeURL()` method:

- `Item.jsp`
- `OrderItem.jsp`
- `Search.jsp`

All embedded URLs for the online bookstore are encoded in ASCII and do not need to be encoded. The `encodeURL()` method is called to illustrate the concept of encoding URLs.

Formatting HTML Pages in the World-of-Books Demo

The online bookstore uses the following locale-sensitive text formatting elements for HTML pages:

- Font family
- Writing direction
- Text alignment

To support multiple locales simultaneously, the online bookstore externalizes these elements to a locale-specific cascading style sheet (CSS) instead of hard-coding them in JSPs. The CSS file structure is the same as the static HTML file structure for the WOB online help.

The CSS files are as follows:

- `$WOB_HOME/docroot/wob/css/style.css` (the default CSS)
- `$WOB_HOME/docroot/wob/css/ar/style.css`
- `$WOB_HOME/docroot/wob/css/he/style.css`
- `$WOB_HOME/docroot/wob/css/iw/style.css`
- `$WOB_HOME/docroot/wob/css/ja/style.css`
- `$WOB_HOME/docroot/wob/css/zh/style.css`

In `$WOB_HOME/docroot/wob/jsp/header.jsp`, the `getLocalizedURL()` method of the `Localizer` bean gets the full path of the CSS that corresponds to the current locale. If there is no CSS that is specific to the locale, then the application uses the default CSS.

The following is the CSS for Arabic text:

```
html { direction: rtl }
h3 { font-size: 100%;
     text-align: end;
     font-weight: bold;
     color: #FFFFFF }
```

The Arabic CSS defines the writing direction of the HTML page as right to left (RTL). The text is always aligned to the end of the writing direction.

The following is the CSS for Japanese text:

```
html { direction: ltr }
h3 { font-size: 100%;
     text-align: end;
     font-family: "MS Gothic", "MS Mincho", "Times New Roman"…
     font-weight: bold;
     color: #FFFFFF }
tr { font-family: "MS Gothic", "MS Mincho", "Times New Roman",…
     font-size: 12pt; }
p {  font-family: "MS Gothic", "MS Mincho" "Times New Roman",…
     font-size: 12pt}
```

The Japanese CSS defines the writing direction as left to right (LTR). The text is aligned to the end of the writing direction. The font families that are used for displaying Japanese text are MS Gothic and MS Mincho. These are Japanese Microsoft Windows fonts. If you do not specify the font family in the CSS, then the application uses the default font of the browser.

Accessing the Database in the World-of-Books Demo

The WOB demo uses the Oracle JDBC driver to access an Oracle9*i* database. The JDBC driver transparently converts the data stored in the database to and from Java strings. No special handling is necessary to access Unicode data stored in the database in most cases.

There is one case in which you need special data handling. When a Java string is bound to a column of the `NVARCHAR` datatype in an `INSERT` or `UPDATE` SQL statement, you should call the `setFormOfUse()` method of the `OraclePreparedStatement` class to tell JDBC that the target column is of the `NVARCHAR` datatype. The `setFormOfUse()` method is called in `$WOB_HOME/src/orcalc/demo/wob2/supp/beans/insertItem.java` when a new book is inserted into the `books` table.

Organizing the Content of HTML Pages in the World-of-Books Demo

The online bookstore consists of the following translatable content:

- Online help as static HTML and image files
- Strings or messages stored for use in composing an HTML page

- Dynamic book information such as the book name and author

This section contains the following topics:

- Static Files for World-of-Books Online Help
- Using Resource Bundles for the Content of World-of-Books HTML Pages

Static Files for World-of-Books Online Help

The static HTML files for the WOB online help are located in `$WOB_HOME/docroot/wob/help`. The English version of the online help is stored at the top level of the `help` directory. The translated help for each locale is stored in the corresponding `help/`*`locale_name`* directory. For example, the Japanese online help is stored in the `help/ja_JP` directory.

The current user's locale determines which help subdirectory the application uses. The `Localizer` bean stores the user's current locale. The `getLocalizedURL()` method returns the correct path of an HTML file that corresponds to the user's locale. Given the relative help path of `../help/index.html` and the current locale of `ja_JP`, this method checks for existence of the following files in the order they are listed and returns the first one it finds:

- `$WOB_HOME/docroot/wob/help/ja_JP/index.html`
- `$WOB_HOME/docroot/wob/help/ja/index.html`
- `$WOB_HOME/docroot/wob/help/index.html`

The `header.jsp` file calls this method to get the correct path for every translated HTML file and uses the result to construct the `HREF` tag to reference the appropriate online help.

Using Resource Bundles for the Content of World-of-Books HTML Pages

A list resource bundle stores all translatable messages that comprise the online bookstore user interface. The resource bundle is located in `$WOB_HOME/src/oracle/demo/wob2/wob/resource/MessageBundle.java`. This resource bundle is translated into 27 languages, and the translated resource bundle names have suffixes that correspond to the Java locale name.

The `getMessage()` method of the `Localizer` bean gets a translated message from the resource bundle that corresponds to the current locale. Most JSPs call this method.

A

Oracle Application Server Translated Languages

The following languages are those into which Oracle Application Server is translated. Oracle Application Server provides runtime support for more languages than those into which Oracle Application Server itself is translated. For a list of all supported languages, see the *Oracle9i Globalization Support Guide* in the Oracle Database Documentation Library.

Table A–1 Translated Languages and Abbreviations

Language	Oracle Language Abbreviation
ARABIC	ar
BRAZILIAN PORTUGUESE	ptb
CANADIAN FRENCH	frc
CATALAN	ca
CZECH	cs
DANISH	dk
DUTCH	nl
FINNISH	sf
FRENCH	f
GERMAN	d
GREEK	el
HEBREW	iw

Table A–1 Translated Languages and Abbreviations (Cont.)

Language	Oracle Language Abbreviation
HUNGARIAN	hu
ITALIAN	i
JAPANESE	ja
KOREAN	ko
LATIN AMERICAN SPANISH	esa
NORWEGIAN	n
POLISH	pl
PORTUGUESE	pt
ROMANIAN	ro
RUSSIAN	ru
SIMPLIFIED CHINESE	zhs
SLOVAK	sk
SPANISH	e
SWEDISH	s
THAI	th
TRADITIONAL CHINESE	zht
TURKISH	tr

Glossary

character set

Defines the binary values that are associated with the characters that make up a language. For example, you can use the ISO-8859-1 character set to encode most Western European languages.

database access descriptor (DAD)

Describes the connect string and Oracle parameters of a target database to which an Oracle HTTP Server `mod_plsql` module connects.

encoding

The character set used in a particular programming environment for the locale to which an Internet application is serving. *See* **page encoding**, **character set**.

locale

Refers to a language and the region (territory) in which the language is spoken. Information about the region includes formats for dates and currency.

page encoding

The character set an HTML page uses for the locale to which an Internet application is serving.

Unicode

A universal character set that defines binary values for characters in almost all languages. Unicode characters can be encoded in 1 to 4 bytes in the UTF-8 character set, in 2 to 4 bytes in the UTF-16 character set, and in 4 bytes in the UTF-32 character set.

Index

A

accessing the database server, 4-1
Albany WT J font, 2-12
ALTER SESSION statement
  in monolingual applications, 2-8
  in multilingual applications, 2-8
Apache::Util module, 3-19
application design, 1-2
architecture
  monolingual, 1-2
  multilingual, 1-2

B

BC4J, configuring for multilingual support, 5-16
bidirectional languages
  formatting HTML pages, 3-20
Business Components for Java (BC4J), configuring for multilingual support, 5-16

C

cascading style sheets, 3-20
C/C++
  database access, 4-5
  database access in multilingual applications, 4-6
  translatable strings, 3-26
C/C++ runtime, configuring, 5-11
CHAR datatypes, 5-17
character set, definition, 1-2
CharEncoding attribute, 3-13
charset argument, 3-19
charset parameter, 3-4
configuration files
  editing manually, 5-3
configuring NLS_LANG
  in Oracle HTTP Server files, 5-4
  on Windows platforms, 5-6
configuring Oracle HTTP Server for multilingual support, 5-3
configuring OracleAS Portal for multilingual support, 5-12
configuring the NLS_LANG environment variable, 5-4
configuring transfer mode for mod_plsql, 5-9
Content-Type HTTP header, 3-4
CREATE DATABASE statement, 5-17

D

database
  centralized, 4-1
  configuring, 4-1
database access
  C/C++, 4-5
  Java, 4-2
  JDBC, 4-2
  multilingual non-Java applications, 4-2
  OCI API, 4-6
  Perl, 4-4
  PL/SQL, 4-3
  Unicode API, 4-7
  Unicode bind and define in Pro*C/C++, 4-8
  World-of-Books demo, 6-20

database character set
setting in the CREATE DATABASE statement, 5-17
database server
accessing, 4-1
decoding HTTP headers, 3-16
in OracleAS Single Sign-On, 3-16
decoding string-type mobile context information headers, 3-17
demo
installing, 6-7
See World-of-Books demo, 6-1
determining user locale
monolingual applications, 2-3
multilingual applications, 2-3
developing locale awareness, 2-1
development environments, 1-6
Discoverer
configuring Java Plus for multilingual support, 5-16
locale awareness, 2-14
doGet() function, 3-5
dynamic environment switching, 5-16

E

editing configuration files, 5-3
encoding
UTF-16, 4-1
UTF-32, 4-1
UTF-8, 4-1
encoding HTML pages, 3-1
encoding URLs, 3-17
Java, 3-18
Perl, 3-19
PL/SQL, 3-19
World-of-Books demo, 6-18
entities
named and numbered, 3-12
environment switching, 5-16
ESCAPE() function, 3-19
escape_uri() function, 3-19

F

fonts
specifying in HTML pages, 3-20
Forms Services
configuring for multilingual support, 5-14
locale awareness, 2-9
locale awareness in a monolingual application, 2-10
locale awareness in a multilingual application, 2-10
Forms servlet, 2-11
formsweb.cfg file, 2-11
from_utf8() function, 3-8

G

GET requests, 3-11
getDateTimeInstance() method, 2-5
getParameter() function, 3-13
getWriter() method, 3-5

H

HTML form input
encoding, 3-11
Java, 3-12
named and numbered entities, 3-12
Perl, 3-14
Perl in multilingual applications, 3-14
PL/SQL, 3-13
PL/SQL monolingual applications, 3-14
PL/SQL multilingual applications, 3-14
World-of-Books demo, 6-17

HTML page encoding
choosing for monolingual applications, 3-2
choosing for multilingual applications, 3-3
in PL/SQL and PSPs, monolingual environments, 3-6
in PL/SQL and PSPs, multilingual environments, 3-6
named and numbered entities, 3-12
specifying, 3-3
specifying in Java servlets and Java Server Pages, 3-4
specifying in OracleAS Mobile Services, 3-8
specifying in OracleAS Web Cache enabled applications, 3-9
specifying in Perl, 3-7
specifying in Perl for monolingual applications, 3-7
specifying in Perl for multilingual applications, 3-7
specifying in PL/SQL and PL/SQL Server Pages, 3-5
specifying in the HTML page header, 3-4
specifying in the HTTP header, 3-4
World-of-Books demo, 6-17
HTML pages
concatenating strings, 3-22
embedding text into images, 3-22
fallback mechanism for translation, 3-22
formatting for bidirectional languages, 3-20
formatting in World-of-Books demo, 6-19
formatting to accommodate text in different languages, 3-20
JavaScript code, 3-22
organizing content for translation, 3-21
organizing static files for translation, 3-22
space for dynamic text, 3-21
specifying fonts, 3-20
translatable C/C++ and Perl strings, 3-26
translatable dynamic content in application schema, 3-28
translatable strings in message tables, 3-27
translation guidelines, 3-21
user interface strings, 3-21
HTTP Content-Type header, 3-7
HTTP headers
decoding, 3-16
decoding in OracleAS Single Sign-On, 3-16
HttpServletRequest.getParameter() API, 3-12

I

IANA encoding names for commonly used locales, 3-2
installing the World-of-Books demo, 6-7

J

Java
accessing the database, 4-2
encoding URLs, 3-18
HTML form input, 3-12
organizing translatable static strings, 3-23
Java encoding names for commonly used locales, 3-2
Java Server Pages
specifying HTML page encoding, 3-4
Java servlets
specifying HTML page encoding, 3-4
JDBC
database access, 4-2

L

LANG environment variable, 5-10, 5-11
languages
OracleAS translated languages, A-1
LC_ALL environment variable, 2-6, 5-10, 5-11

locale
as ISO standard, 2-1
as Java locale object, 2-1
as NLS_LANGUAGE and NLS_TERRITORY parameters, 2-2
as POSIX locale name, 2-2
based on the default ISO locale of the user's browser, 2-4
changing operating system locale, 2-6
common representations, 2-2
definition, 1-2
determined by user input, 2-4
using user profile information from an LDAP directory server, 2-4
locale awareness
C++ applications, 2-6
developing, 2-1
in multilingual Perl and C/C++ applications, 3-1
in OracleAS Discoverer applications, 2-14
Java applications, 2-5
OracleAS Forms Services, 2-9
OracleAS Reports Services, 2-12
OracleAS Wireless Services, 2-9
Perl applications, 2-6
PL/SQL applications, 2-7
SQL applications, 2-7
World-of-Books demo, 6-12
determining locale, 6-13
localizer methods, 6-14
Locale.setDefault() method, 2-5
localizer methods, World-of-Books demo, 6-14

M

manually editing configuration files, 5-3
message tables
translatable strings, 3-27
Mobile Services
specifying HTML page encoding, 3-8
mod_jserv runtime for Java, configuring, 5-10
mod_perl environment, 3-7
mod_perl runtime for Perl scripts, configuring, 5-11
mod_plsql
configuring transfer mode, 5-9
mod_plsql module
datatypes, 3-13
HTML form input in monolingual applications, 3-13
mod_plsql runtime for PL/SQL and PL/SQL Server Pages, configuring, 5-11
monolingual applications
advantages, 1-4
architecture, 1-3
determining user locale, 2-3
disadvantages, 1-4
multilingual applications
advantages, 1-6
architecture, 1-4
database access with C/C++, 4-6
database access with Perl, 4-5
database access with Unicode API, 4-7
database access with Unicode bind and define in Pro*C/C++, 4-8
determining user locale, 2-3
based on ISO locale, 2-4
based on user input, 2-4
based on user profile, 2-4
disadvantages, 1-6
HTML form input in Perl, 3-14

N

native encodings for commonly used locales, 3-2
native2ascii utility, 3-23
NCHAR datatypes, 5-17
NLS_LANG parameter, 2-7
configuring, 5-4
configuring in Oracle HTTP Server files, 5-4
configuring on Windows platforms, 5-6
setting in a multilingual application architecture, 5-8
values for commonly used locales, 5-7

O

OC4J Java runtime, configuring, 5-11
OCI API
  database access, 4-6
  Unicode API, 4-7
Oracle Business Components for Java, 5-16
Oracle character set names for commonly used locales, 3-2
OracleAS Discoverer
  configuring Java Plus for multilingual support, 5-16
  locale awareness, 2-14
OracleAS Forms Services
  configuring for multilingual support, 5-14
  locale awareness, 2-9
OracleAS Infrastructure
  and global deployment, 5-1
OracleAS Mobile Services
  specifying HTML page encoding, 3-8
OracleAS Portal
  configuring for multilingual support, 5-12
OracleAS Reports Services
  locale awareness, 2-12
OracleAS Single Sign-On
  configuring for multilingual support, 5-13
OracleAS Web Cache
  specifying HTML page encoding in Web Cache enabled applications, 3-9
OracleAS Wireless
  configuring encoding for outgoing messages, 5-13
  configuring for multilingual support, 5-13

P

Perl
  database access, 4-4
  database access in multilingual applications, 4-5
  encoding URLs, 3-19
  HTML form input, 3-14
  HTML form input in multilingual applications, 3-14
  specifying HTML page encoding, 3-7
  specifying HTML page encoding for monolingual applications, 3-7
  specifying HTML page encoding for multilingual applications, 3-7
  translatable strings, 3-26
PL/SQL
  database access, 4-3
  encoding URLs, 3-19
  HTML form input, 3-13
  HTML form input in monolingual applications, 3-14
  HTML form input in multilingual applications, 3-14
PL/SQL and PL/SQL Server Pages
  specifying HTML page encoding, 3-5
Portal
  configuring for multilingual support, 5-12
POSIX locale names, 5-10
POST requests, 3-11
Pro*C/C++
  database access, 4-8
programming languages
  supported, 1-6

R

Reports Server
  configuring for multilingual support, 5-15
Reports Services
  locale awareness, 2-12
  locale awareness in a multilingual application, 2-13
  page encoding in HTML output, 3-10
  page encoding in XML output, 3-11
  specifying the page encoding, 3-10
runtime default locale, configuring in a monolingual application architecture, 5-10

S

schema
  translatable content, 3-28
Servlet API, 3-12
setContentType() method, 3-5
setlocale() function
  monolingual applications, 2-6
  multilingual applications, 2-6
setting NLS_LANG
  monolingual applications, 5-7
setting NLS_LANG parameter
  in a multilingual application architecture, 5-8
Single Sign-On
  configuring for multilingual support, 5-13
String.getBytes() method, 2-5
String.getBytes(String encoding) method, 2-5
string-type mobile context information headers
  decoding, 3-17
strlen() function, 4-7
switching environments, 5-16

T

text datatypes, 4-7
to_utf8() function, 3-15
transfer mode
  configuring for mod_plsql, 5-9
translated languages, A-1
translation
  organizing HTML page content, 3-21

U

Unicode
  definition, 1-2
Unicode API
  database access, 4-7
Unicode bind and define
  database access, 4-8
Unicode data
  storing in the database, 5-17
UNICODE::MAPUTF8 Perl module, 3-7
url argument, 3-19
URLs
  encoding, 3-17
  encoding in Java, 3-18
  encoding in Perl, 3-19
  encoding in PL/SQL, 3-19
  encoding in World-of-Books demo, 6-18
  with embedded query strings, 3-17
utext datatype, 4-7, 4-8
UTF-16 encoding, 4-1
UTF-32 encoding, 4-1
UTF-8 encoding, 3-14, 4-1
  for HTML pages, 3-3
  limitations with Netscape 4.x browser, 3-3
UTL_URL package, 3-19
uvarchar datatype, 4-8

W

wcslen() function, 4-7
Web Toolkit API, 3-6
Wireless
  configuring encoding for outgoing messages, 5-13
  configuring for multilingual support, 5-13

World-of-Books demo
architecture, 6-2
building, 6-9
database access, 6-20
deploying, 6-10
design, 6-4
directory structure, 6-8
encoding URLs, 6-18
formatting HTML pages, 6-19
HTML form input, 6-17
HTML page encoding, 6-17
installing, 6-7
locale awareness, 6-12
determining locale, 6-13
localizer methods, 6-14
online help, 6-21
organizing HTML content, 6-20
organizing static files, 6-21
overview, 6-1
resource bundles, 6-21
running, 6-11
schema design, 6-5
books table, 6-5
customers table, 6-5
docs table (book content), 6-6
searching book contents, 6-16
sorting query results, 6-15
source file location, 6-7