The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent and other intellectual and industrial property laws. Reverse engineering, disassembly or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

**Restricted Rights Notice** Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee’s responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Oracle Corporation disclaims liability for any damages caused by such use of the Programs.

Oracle is a registered trademark, and Oracle Store, Oracle8, Oracle8i, Oracle9i, SQL*Plus, PL/SQL, Pro*C/C++, and Pro*Cobol are trademarks or registered trademarks of Oracle Corporation. Other names may be trademarks of their respective owners.
## Contents

Send Us Your Comments ........................................................................................................... xv

Preface ........................................................................................................................................ xvii

What's New in Globalization Support ? ................................................................................... xxvii

1 Overview of Globalization Support

Globalization Support Architecture ........................................................................................... 1-2
   Locale Data on Demand ........................................................................................................ 1-2
   Architecture to Support Multilingual Applications ......................................................... 1-3
   Using Unicode in a Multilingual Database ......................................................................... 1-5

Globalization Support Features .............................................................................................. 1-5
   Language Support ............................................................................................................... 1-6
   Territory Support ................................................................................................................ 1-7
   Date and Time Formats ....................................................................................................... 1-7
   Monetary and Numeric Formats ......................................................................................... 1-7
   Calendars Feature ............................................................................................................... 1-8
   Linguistic Sorting ................................................................................................................ 1-8
   Character Set Support ......................................................................................................... 1-8
   Character Semantics ........................................................................................................... 1-8
   Customization of Locale and Calendar Data ...................................................................... 1-9
   Unicode Support ................................................................................................................ 1-9
2 Choosing a Character Set

Character Set Encoding .................................................................................................................. 2-2
What is an Encoded Character Set? .............................................................................................. 2-2
Which Characters Are Encoded? .................................................................................................... 2-3
What Characters Does a Character Set Support? ....................................................................... 2-5
How are Characters Encoded? ...................................................................................................... 2-9
Naming Convention for Oracle Character Sets ........................................................................... 2-11

Length Semantics .......................................................................................................................... 2-12

Choosing an Oracle Database Character Set ............................................................................. 2-14
Current and Future Language Requirements .............................................................................. 2-16
Client Operating System and Application Compatibility ............................................................ 2-16
Character Set Conversion Between Clients and the Server ........................................................... 2-17
Performance Implications of Choosing a Database Character Set ............................................. 2-17
Restrictions on Database Character Sets ..................................................................................... 2-18
Choosing a National Character Set .............................................................................................. 2-19
Summary of Supported Datatypes ............................................................................................... 2-19

Changing the Character Set After Database Creation ................................................................. 2-20
Monolingual Database Scenario ................................................................................................... 2-20
Character Set Conversion in a Monolingual Scenario ................................................................. 2-21

Multilingual Database Scenarios .................................................................................................. 2-23
Restricted Multilingual Support ................................................................................................... 2-24
Unrestricted Multilingual Support ............................................................................................... 2-25

3 Setting Up a Globalization Support Environment

Setting NLS Parameters .............................................................................................................. 3-2
Choosing a Locale with the NLS_LANG Environment Variable ................................................... 3-5
Specifying the Value of NLS_LANG .............................................................................................. 3-7
Overriding Language and Territory Specifications ....................................................................... 3-8
Locale Variants .............................................................................................................................. 3-9
Should the NLS_LANG Setting Match the Database Character Set? ........................................... 3-10

NLS Database Parameters .......................................................................................................... 3-11
NLS Data Dictionary Views .......................................................................................................... 3-11
NLS Dynamic Performance Views ............................................................................................... 3-11
OCINlsGetInfo() Function .......................................................................................................... 3-12

Language and Territory Parameters ............................................................................................. 3-12
4 Datetime Datatypes and Time Zone Support

Overview of Datetime and Interval Datatypes and Time Zone Support ........................................... 4-2
Datetime and Interval Datatypes ........................................................................................................ 4-2
  Datetime Datatypes .......................................................................................................................... 4-3
  Interval Datatypes ........................................................................................................................... 4-12
Datetime and Interval Arithmetic and Comparisons ........................................................................ 4-14
Datet ime and Interval Arithmetic ................................................................. 4-14
Datet ime Comparisons ............................................................................. 4-15
Explicit Conversion of Datetime Datatypes ............................................. 4-16
Datet ime SQL Functions ........................................................................... 4-16
Datet ime and Time Zone Parameters and Environment Variables ........ 4-18
Datet ime Format Parameters ................................................................... 4-18
Time Zone Environment Variables ............................................................ 4-19
Daylight Saving Time Session Parameter .................................................. 4-20
Choosing a Time Zone File ......................................................................... 4-20
Setting the Database Time Zone ................................................................. 4-23
Setting the Session Time Zone ................................................................. 4-24
Converting Time Zones With the AT TIME ZONE Clause ....................... 4-25
Support for Daylight Saving Time ............................................................... 4-26
Examples: The Effect of Daylight Saving Time on Datetime Calculations 4-26

5 Linguistic Sorting and String Searching
Overview of Oracle's Sorting Capabilities ................................................ 5-2
Using Binary Sorts ..................................................................................... 5-2
Using Linguistic Sorts ............................................................................... 5-3
Monolingual Linguistic Sorts ...................................................................... 5-3
Multilingual Linguistic Sorts ..................................................................... 5-4
Multilingual Sorting Levels ....................................................................... 5-5
Linguistic Sort Features ............................................................................. 5-7
Base Letters ............................................................................................... 5-7
Ignorable Characters .................................................................................. 5-7
Contracting Characters ............................................................................... 5-8
Expanding Characters ............................................................................... 5-8
Context-Sensitive Characters .................................................................... 5-8
Canonical Equivalence ............................................................................... 5-9
Reverse Secondary Sorting ...................................................................... 5-10
Character Rearrangement for Thai and Laotian Characters ....................... 5-10
Special Letters ......................................................................................... 5-10
Special Combination Letters ..................................................................... 5-10
Special Uppercase Letters ....................................................................... 5-11
Special Lowercase Letters ....................................................................... 5-11
Case-Insensitive and Accent-Insensitive Linguistic Sorts .................................................... 5-11
Examples of Case-Insensitive and Accent-Insensitive Sorts ................................................ 5-14
Specifying a Case-Insensitive or Accent-Insensitive Sort ................................................... 5-15
Linguistic Sort Examples ........................................................................................................ 5-17
Using Linguistic Indexes ....................................................................................................... 5-19
Linguistic Indexes for Multiple Languages ........................................................................ 5-20
Requirements for Using Linguistic Indexes ...................................................................... 5-20
Searching Linguistic Strings .............................................................................................. 5-22
SQL Regular Expressions in a Multilingual Environment .................................................. 5-22
Character Range '[x-y]' in Regular Expressions ................................................................... 5-23
Collation Element Delimiter '[ . ]' in Regular Expressions .............................................. 5-23
Character Class '[: :]' in Regular Expressions .................................................................... 5-24
Equivalence Class '[ = = ]' in Regular Expressions .......................................................... 5-24
Examples: Regular Expressions ......................................................................................... 5-24

6 Supporting Multilingual Databases with Unicode

Overview of Unicode ........................................................................................................... 6-2
What is Unicode? .................................................................................................................. 6-2
  Supplementary Characters .............................................................................................. 6-3
  Unicode Encodings ......................................................................................................... 6-3
  Oracle’s Support for Unicode ........................................................................................ 6-5
Implementing a Unicode Solution in the Database .......................................................... 6-6
  Enabling Multilingual Support with Unicode Databases .............................................. 6-7
  Enabling Multilingual Support with Unicode Datatypes .............................................. 6-8
  How to Choose Between a Unicode Database and a Unicode Datatype Solution ........ 6-10
  Comparing Unicode Character Sets for Database and Datatype Solutions ................. 6-12
Unicode Case Studies ......................................................................................................... 6-15
Designing Database Schemas to Support Multiple Languages ......................................... 6-17
  Specifying Column Lengths for Multilingual Data ....................................................... 6-18
  Storing Data in Multiple Languages ............................................................................ 6-18
  Storing Documents in Multiple Languages in LOB Datatypes ..................................... 6-20
  Creating Indexes for Searching Multilingual Document Contents .............................. 6-21

7 Programming with Unicode

Overview of Programming with Unicode ........................................................................... 7-2
Database Access Product Stack and Unicode ................................................................. 7-2
SQL and PL/SQL Programming with Unicode ............................................................... 7-4
  SQL NCHAR Datatypes ............................................................................................... 7-5
  Implicit Datatype Conversion Between NCHAR and Other Datatypes .................... 7-7
  Exception Handling for Data Loss During Datatype Conversion .............................. 7-7
  Rules for Implicit Datatype Conversion ..................................................................... 7-8
  SQL Functions for Unicode Datatypes ................................................................. 7-9
  Other SQL Functions ............................................................................................... 7-11
  Unicode String Literals ............................................................................................ 7-11
  Using the UTL_FILE Package with NCHAR Data .................................................... 7-12
OCI Programming with Unicode ................................................................................. 7-13
  OCIEnvNlsCreate() Function for Unicode Programming ........................................ 7-14
  OCI Unicode Code Conversion .................................................................................. 7-15
  When the NLS_LANG Character Set is UTF8 or AL32UTF8 in OCI ......................... 7-18
  Binding and Defining SQL CHAR Datatypes in OCI ............................................... 7-19
  Binding and Defining SQL NCHAR Datatypes in OCI ............................................. 7-20
  Binding and Defining CLOB and NCLOB Unicode Data in OCI .............................. 7-21
Pro*C/C++ Programming with Unicode ....................................................................... 7-22
  Pro*C/C++ Data Conversion in Unicode .................................................................. 7-22
  Using the VARCHAR Datatype in Pro*C/C++ ............................................................. 7-23
  Using the NVARCHAR Datatype in Pro*C/C++ ......................................................... 7-24
  Using the UVARCHAR Datatype in Pro*C/C++ ......................................................... 7-24
JDBC Programming with Unicode ................................................................................ 7-25
  Binding and Defining Java Strings to SQL CHAR Datatypes .................................... 7-26
  Binding and Defining Java Strings to SQL NCHAR Datatypes ................................. 7-27
  Using the SQL NCHAR Datatypes Without Changing the Code .............................. 7-28
  Data Conversion in JDBC ....................................................................................... 7-29
  Using oracle.sql.CHAR in Oracle Object Types ..................................................... 7-31
  Restrictions on Accessing SQL CHAR Data with JDBC ........................................ 7-34
ODBC and OLE DB Programming with Unicode ......................................................... 7-37
  Unicode-Enabled Drivers in ODBC and OLE DB ...................................................... 7-37
  OCI Dependency in Unicode .................................................................................... 7-37
  ODBC and OLE DB Code Conversion in Unicode .................................................. 7-38
  ODBC Unicode Datatypes ....................................................................................... 7-40
  OLE DB Unicode Datatypes .................................................................................... 7-40
8 Oracle Globalization Development Kit

Overview of the Oracle Globalization Development Kit .......................................................... 8-2
Designing a Global Internet Application ................................................................................. 8-2
  Deploying a Monolingual Internet Application .................................................................. 8-3
  Deploying a Multilingual Internet Application ................................................................ 8-5
Developing a Global Internet Application ............................................................................. 8-7
  Locale Determination ........................................................................................................ 8-8
  Locale Awareness .............................................................................................................. 8-9
  Localizing the Content ..................................................................................................... 8-10
Getting Started with the Globalization Development Kit ..................................................... 8-10
GDK Application Framework for J2EE .................................................................................. 8-12
  Making the GDK Framework Available to J2EE Applications ........................................ 8-15
  Integrating Locale Sources into the GDK Framework ..................................................... 8-16
  Getting the User Locale From the GDK Framework ....................................................... 8-17
  Implementing Locale Awareness Using the GDK Localizer ............................................. 8-19
  Defining the Supported Application Locales in the GDK ............................................. 8-20
  Handling Non-ASCII Input and Output in the GDK Framework ..................................... 8-21
  Managing Localized Content in the GDK ..................................................................... 8-24
GDK Java API .......................................................................................................................... 8-26
  Oracle Locale Information in the GDK ........................................................................... 8-27
  Oracle Locale Mapping in the GDK .................................................................................. 8-28
  Oracle Character Set Conversion (JDK 1.4 and Later) in the GDK ................................. 8-29
  Oracle Date, Number, and Monetary Formats in the GDK ............................................. 8-30
  Oracle Binary and Linguistic Sorts in the GDK ............................................................... 8-31
  Oracle Language and Character Set Detection in the GDK ........................................... 8-32
  Oracle Translated Locale and Time Zone Names in the GDK ....................................... 8-34
  Using the GDK for E-Mail Programs ............................................................................. 8-35
The GDK Application Configuration File ............................................................................... 8-36
  locale-charset-map ......................................................................................................... 8-37
**9 SQL and PL/SQL Programming in a Global Environment**

Locale-Dependent SQL Functions with Optional NLS Parameters ........................................ 9-2
  Default Values for NLS Parameters in SQL Functions...................................................... 9-3
  Specifying NLS Parameters in SQL Functions ................................................................... 9-3
  Unacceptable NLS Parameters in SQL Functions .............................................................. 9-5

**Other Locale-Dependent SQL Functions** ........................................................................ 9-5
  The CONVERT Function .................................................................................................... 9-6
  SQL Functions for Different Length Semantics ................................................................. 9-6
  LIKE Conditions for Different Length Semantics ............................................................ 9-8
  Character Set SQL Functions ............................................................................................ 9-8
  The NLSSORT Function .................................................................................................... 9-9

**Miscellaneous Topics for SQL and PL/SQL Programming in a Global Environment** ...... 9-13
  SQL Date Format Masks ................................................................................................... 9-13
  Calculating Week Numbers ............................................................................................... 9-13
  SQL Numeric Format Masks ............................................................................................. 9-14
  Loading External BFILE Data into LOB Columns ............................................................. 9-14

**10 OCI Programming in a Global Environment**

Using the OCI NLS Functions ............................................................................................. 10-2
Specifying Character Sets in OCI ................................................................. 10-2
Getting Locale Information in OCI ............................................................ 10-3
Mapping Locale Information Between Oracle and Other Standards ....... 10-4
Manipulating Strings in OCI ................................................................. 10-4
Classifying Characters in OCI ............................................................. 10-7
Converting Character Sets in OCI .......................................................... 10-7
OCI Messaging Functions ................................................................. 10-8
lmsgen Utility ......................................................................................... 10-8

11 Character Set Migration
Overview of Character Set Migration ................................................ 11-2
Data Truncation ...................................................................................... 11-2
Character Set Conversion Issues ............................................................ 11-4
Changing the Database Character Set of an Existing Database ....... 11-7
Migrating Character Data Using a Full Export and Import ................. 11-8
Migrating a Character Set Using the CSALTER Script ....................... 11-8
Migrating Character Data Using the CSALTER Script and Selective Imports .. 11-10
Migrating to NCHAR Datatypes ............................................................ 11-10
Migrating Version 8 NCHAR Columns to Oracle9i and Later .......... 11-11
Changing the National Character Set .................................................. 11-12
Migrating CHAR Columns to NCHAR Columns .................................. 11-12
Tasks to Recover Database Schema After Character Set Migration ...... 11-16

12 Character Set Scanner Utilities
The Language and Character Set File Scanner .................................... 12-2
Syntax of the LCSSCAN Command ....................................................... 12-2
Examples: Using the LCSSCAN Command ............................................ 12-4
Getting Command-Line Help for the Language and Character Set File Scanner ..................................................................... 12-4
Supported Languages and Character Sets ........................................... 12-5
LCSSCAN Error Messages ................................................................. 12-5
The Database Character Set Scanner .................................................. 12-6
Conversion Tests on Character Data ..................................................... 12-7
Scan Modes in the Database Character Set Scanner .......................... 12-7
Full Database Scan .............................................................................. 12-7
User Scan ............................................................................................ 12-8
Oracle Locale Builder Windows and Dialog Boxes ............................................................... 13-4
Creating a New Language Definition with the Oracle Locale Builder ................................. 13-8
Creating a New Territory Definition with the Oracle Locale Builder ................................. 13-11
Customizing Time Zone Data ............................................................................................. 13-17
Customizing Calendars with the NLS Calendar Utility ......................................................... 13-17
Displaying a Code Chart with the Oracle Locale Builder .................................................... 13-18
Creating a New Character Set Definition with the Oracle Locale Builder ......................... 13-22
Character Sets with User-Defined Characters .................................................................... 13-23
Oracle Character Set Conversion Architecture .................................................................... 13-24
Unicode 3.2 Private Use Area .............................................................................................. 13-25
User-Defined Character Cross-References Between Character Sets .................................. 13-25
Guidelines for Creating a New Character Set from an Existing Character Set .................. 13-25
Example: Creating a New Character Set Definition with the Oracle Locale Builder ........ ... 13-26
Creating a New Linguistic Sort with the Oracle Locale Builder ........................................... 13-32
Changing the Sort Order for All Characters with the Same Diacritic ............................... 13-35
Changing the Sort Order for One Character with a Diacritic .............................................. 13-38
Generating and Installing NLB Files ..................................................................................... 13-40
Transportable NLB Data ....................................................................................................... 13-42

A Locale Data

Languages .............................................................................................................................. A-2
Translated Messages ........................................................................................................... A-4
Territories ............................................................................................................................ A-6
Character Sets ..................................................................................................................... A-7
  Asian Language Character Sets ....................................................................................... A-8
  European Language Character Sets .............................................................................. A-10
  Middle Eastern Language Character Sets ..................................................................... A-16
  Universal Character Sets ............................................................................................... A-19
  Character Set Conversion Support .............................................................................. A-19
  Subsets and Supersets ................................................................................................. A-20
Language and Character Set Detection Support ............................................................. A-23
Linguistic Sorts ................................................................................................................... A-25
Calendar Systems ............................................................................................................. A-28
Time Zone Names ............................................................................................................. A-30
Obsolete Locale Data ........................................................................................................ A-37
Updates to the Oracle Language and Territory Definition Files ........................................ A-37
Obsolete Linguistic Sorts ........................................................................................................ A-38
CIS Is No Longer the Default Territory When the Language is RUSSIAN ................. A-38
YUGOSLAVIA Is a Deprecated Territory ......................................................................... A-38
New Names for Obsolete Character Sets ........................................................................ A-38
AL24UTFSS Character Set Desupported ........................................................................ A-40
Bengali Language Definition Deprecated ........................................................................ A-40
Czechoslovakia Territory Definition Deprecated ............................................................ A-40

B  Unicode Character Code Assignments

Unicode Code Ranges ........................................................................................................ B-2
UTF-16 Encoding .............................................................................................................. B-3
UTF-8 Encoding ............................................................................................................... B-3

Glossary

Index
Send Us Your Comments

Oracle Database Globalization Support Guide, 10g Release 1 (10.1)
Part No. B10749-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. 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?

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

- Electronic mail: infodev_us@oracle.com
- FAX: (650) 506-7227   Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op11
  Redwood Shores, CA  94065
  USA

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

If you have problems with the software, please contact your local Oracle Support Services.
This manual describes Oracle globalization support for the database. It explains how to set up a globalization support environment, choose and migrate a character set, customize locale data, do linguistic sorting, program in a global environment, and program with Unicode.

This preface contains these topics:

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

*Oracle Database Globalization Support Guide* is intended for database administrators, system administrators, and database application developers who perform the following tasks:

- Set up a globalization support environment
- Choose, analyze, or migrate character sets
- Sort data linguistically
- Customize locale data
- Write programs in a global environment
- Use Unicode

To use this document, you need to be familiar with relational database concepts, basic Oracle server concepts, and the operating system environment under which you are running Oracle.

Organization

This document contains:

**Chapter 1, "Overview of Globalization Support"**
This chapter contains an overview of globalization and Oracle’s approach to globalization.

**Chapter 2, "Choosing a Character Set"**
This chapter describes how to choose a character set.

**Chapter 3, "Setting Up a Globalization Support Environment"**
This chapter contains sample scenarios for enabling globalization capabilities.

**Chapter 4, "Datetime Datatypes and Time Zone Support"**
This chapter describes Oracle’s datetime and interval datatypes, datetime SQL functions, and time zone support.

**Chapter 5, "Linguistic Sorting and String Searching"**
This chapter describes linguistic sorting.
Chapter 6, "Supporting Multilingual Databases with Unicode"
This chapter describes Unicode considerations for databases.

Chapter 7, "Programming with Unicode"
This chapter describes how to program in a Unicode environment.

Chapter 8, "Oracle Globalization Development Kit"
This chapter describes the Globalization Development Kit.

Chapter 9, "SQL and PL/SQL Programming in a Global Environment"
This chapter describes globalization considerations for SQL programming.

Chapter 10, "OCI Programming in a Global Environment"
This chapter describes globalization considerations for OCI programming.

Chapter 11, "Character Set Migration"
This chapter describes character set conversion issues and character set migration.

Chapter 12, "Character Set Scanner Utilities"
This chapter describes how to use the Character Set Scanner utility to analyze character data.

Chapter 13, "Customizing Locale"
This chapter explains how to use the Oracle Locale Builder utility to customize locales. It also contains information about time zone files and customizing calendar data.

Appendix A, "Locale Data"
This chapter describes the languages, territories, character sets, and other locale data supported by the Oracle server.

Appendix B, "Unicode Character Code Assignments"
This chapter lists Unicode code point values.

Glossary
The glossary contains definitions of globalization support terms.
Related Documentation

For more information, see these Oracle resources:

Many of the examples in this book use the sample schemas of the seed database, which is installed by default when you install Oracle. Refer to Oracle Database Sample Schemas for information on how these schemas were created and how you can use them yourself.

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/index.html

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/index.html

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 for Windows Operating Systems

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.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bold</td>
<td>Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.</td>
<td>When you specify this clause, you create an index-organized table.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Italics</strong></td>
<td>Italic typeface indicates book titles or emphasis.</td>
<td><em>Oracle Database Concepts</em></td>
</tr>
<tr>
<td><strong>UPPERCASE</strong></td>
<td>Uppercase monospace typeface indicates elements supplied by the system.</td>
<td>You can specify this clause only for a NUMBER column.</td>
</tr>
<tr>
<td><strong>monospace</strong></td>
<td>Uppercase monospace typeface indicates elements supplied by the system.</td>
<td>You can back up the database by using the BACKUP command.</td>
</tr>
<tr>
<td><strong>(fixed-width)</strong></td>
<td>Uppercase monospace typeface indicates elements supplied by the system.</td>
<td>Query the TABLE_NAME column in the USER_ TABLES data dictionary view.</td>
</tr>
<tr>
<td><strong>font</strong></td>
<td>Uppercase monospace typeface indicates elements supplied by the system.</td>
<td>Use the DBMS_STATS_GENERATE_STATS procedure.</td>
</tr>
<tr>
<td><strong>lowercase</strong></td>
<td>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.</td>
<td>Enter sqlplus to open SQL*Plus.</td>
</tr>
<tr>
<td><strong>monospace</strong></td>
<td>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.</td>
<td>The password is specified in the orapwd file.</td>
</tr>
<tr>
<td><strong>(fixed-width)</strong></td>
<td>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.</td>
<td>Back up the datafiles and control files in the /disk1/oracle/dbs directory.</td>
</tr>
<tr>
<td><strong>font</strong></td>
<td>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.</td>
<td>The department_id, department_name, and location_id columns are in the hr.departments table.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.</td>
<td>Set the QUERY_REWRITE_ENABLED initialization parameter to true.</td>
</tr>
<tr>
<td><strong>lowercase</strong></td>
<td>Lowercase monospace font represents placeholders or variables.</td>
<td>Connect as oe user.</td>
</tr>
<tr>
<td><strong>italic</strong></td>
<td>Lowercase italic monospace font represents placeholders or variables.</td>
<td>The JRepUtil class implements these methods.</td>
</tr>
<tr>
<td><strong>monospace</strong></td>
<td>Lowercase italic monospace font represents placeholders or variables.</td>
<td>You can specify the parallel_clause.</td>
</tr>
<tr>
<td><strong>(fixed-width)</strong></td>
<td>Lowercase italic monospace font represents placeholders or variables.</td>
<td>Run Uold_release.SQL where old_release refers to the release you installed prior to upgrading.</td>
</tr>
<tr>
<td><strong>font</strong></td>
<td>Lowercase italic monospace font represents placeholders or variables.</td>
<td>Run Uold_release.SQL where old_release refers to the release you installed prior to upgrading.</td>
</tr>
</tbody>
</table>

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.
Brackets enclose one or more optional items. Do not enter the brackets.

Braces enclose two or more items, one of which is required. Do not enter the braces.

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.

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

Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.

You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.

Italicized text indicates placeholders or variables for which you must supply particular values.

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.
The following table describes conventions for Windows operating systems and provides examples of their use.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>lowercase</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>SELECT last_name, employee_id FROM employees; sqlplus hr/hr CREATE USER mjones IDENTIFIED BY ty3MU9;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conventions for Windows Operating Systems</td>
<td>The following table describes conventions for Windows operating systems and provides examples of their use.</td>
<td>Conventions for Windows Operating Systems</td>
</tr>
</tbody>
</table>
### Convention | Meaning | Example
---|---|---
Special characters | The backslash (\) special character is sometimes required as an escape character for the double quotation mark (" ) special character at the Windows command prompt. Parentheses and the single quotation mark (’) do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters. | C:\>exp scott/tiger TABLES=emp
QUERY="WHERE job='SALESMA' and sal<1600"
C:\>imp SYSTEM/password FROMUSER=scott TABLES=(emp, dept)

**HOME_NAME** | Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore. | C:\> net start OracleHOME_NAMETNSListener

**ORACLE_HOME** and **ORACLE_BASE** | In releases prior to Oracle8i release 8.1.3, when you installed Oracle components, all subdirectories were located under a top level ORACLE_HOME directory. For Windows NT, the default location was C:\orant for Windows NT.

This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level ORACLE_HOME directory. There is a top level directory called ORACLE_BASE that by default is C:\oracle. If you install the latest Oracle release on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is C:\\oracle\\orann, where nn is the latest release number. The Oracle home directory is located directly under ORACLE_BASE.

All directory path examples in this guide follow OFA conventions.

Refer to Oracle Database Platform Guide for Windows for additional information about OFA compliances and for information about installing Oracle products in non-OFA compliant directories.

Go to the ORACLE_BASE\ORACLE_HOME\rdbms\admin directory.
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 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 does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.
What’s New in Globalization Support?

This section describes new features of globalization support and provides pointers to additional information.

Unicode 3.2 Support

This release supports Unicode 3.2.

See Also: Chapter 6, "Supporting Multilingual Databases with Unicode"

Accent-Insensitive and Case-Insensitive Linguistic Sorts and Queries

Oracle provides linguistic sorts and queries that use information about base letter, accents, and case to sort character strings. This release enables you to specify a sort or query on the base letters only (accent-insensitive) or on the base letters and the accents (case-insensitive).

See Also: "Linguistic Sort Features" on page 5-7

Character Set Scanner Utilities Enhancements

The Database Character Set Scanner and Converter now supports object types.

The new LCSD parameter enables the Database Character Set Scanner (CSSCAN) to perform language and character set detection on the data cells categorized by the LCSDATA parameter. The Database Character Set Scanner reports have also been enhanced.
Database Character Set Scanner CSALTER Script
The CSALTER script is a DBA tool for special character set migration.

The Language and Character Set File Scanner Utility
The Language and Character Set File Scanner (LCSSCAN) is a high-performance, statistically based utility for determining the character set and language for unspecified plain file text.

Globalization Development Kit
The Globalization Development Kit (GDK) simplifies the development process and reduces the cost of developing Internet applications that will support a global multilingual market. GDK includes APIs, tools, and documentation that address many of the design, development, and deployment issues encountered in the creation of global applications. GDK lets a single program work with text in any language from anywhere in the world. It enables you to build a complete multilingual server application with little more effort than it takes to build a monolingual server application.

Regular Expressions
This release supports POSIX-compliant regular expressions to enhance search and replace capability in programming environments such as UNIX and Java. In SQL, this new functionality is implemented through new functions that are regular expression extensions to existing SQL functions such as LIKE, REPLACE, and INSTR. This implementation supports multilingual queries and is locale-sensitive.

Displaying Code Charts for Unicode Character Sets
Oracle Locale Builder can display code charts for Unicode character sets.

See Also: Chapter 12, "Character Set Scanner Utilities"
See Also: Chapter 8, "Oracle Globalization Development Kit"
See Also: "SQL Regular Expressions in a Multilingual Environment" on page 5-22
See Also: "Displaying a Code Chart with the Oracle Locale Builder" on page 13-18
Locale Variants

In previous releases, Oracle defined language and territory definitions separately. This resulted in the definition of a territory being independent of the language setting of the user. In this release, some territories can have different date, time, number, and monetary formats based on the language setting of a user. This type of language-dependent territory definition is called a locale variant.

See Also: "Locale Variants" on page 3-9

Transportable NLB Data

NLB files that are generated on one platform can be transported to another platform by, for example, FTP. The transported NLB files can be used the same way as the NLB files that were generated on the original platform. This is convenient because locale data can be modified on one platform and copied to other platforms.

See Also: "Transportable NLB Data" on page 13-42

NLS_LENGTH_SEMANTICS

NLS_LENGTH_SEMANTICS is now supported as an environment variable.

See Also: "NLS_LENGTH_SEMANTICS" on page 3-44

Implicit Conversion Between CLOB and NCLOB Datatypes

Implicit conversion between CLOB and NCLOB datatypes is now supported.

See Also: "Choosing a National Character Set" on page 2-19

Updates to the Oracle Language and Territory Definition Files

Changes have been made to the content in some of the language and territory definition files in Oracle Database 10g.

See Also: "Obsolete Locale Data" on page A-37
This chapter provides an overview of Oracle globalization support. It includes the following topics:

- Globalization Support Architecture
- Globalization Support Features
Globalization Support Architecture

Oracle's globalization support enables you to store, process, and retrieve data in native languages. It ensures that database utilities, error messages, sort order, and date, time, monetary, numeric, and calendar conventions automatically adapt to any native language and locale.

In the past, Oracle's globalization support capabilities were referred to as National Language Support (NLS) features. National Language Support is a subset of globalization support. National Language Support is the ability to choose a national language and store data in a specific character set. Globalization support enables you to develop multilingual applications and software products that can be accessed and run from anywhere in the world simultaneously. An application can render content of the user interface and process data in the native users' languages and locale preferences.

Locale Data on Demand

Oracle's globalization support is implemented with the Oracle NLS Runtime Library (NLSRTL). The NLS Runtime Library provides a comprehensive suite of language-independent functions that allow proper text and character processing and language convention manipulations. Behavior of these functions for a specific language and territory is governed by a set of locale-specific data that is identified and loaded at runtime.

The locale-specific data is structured as independent sets of data for each locale that Oracle supports. The data for a particular locale can be loaded independent of other locale data. The advantages of this design are as follows:

- You can manage memory consumption by choosing the set of locales that you need.
- You can add and customize locale data for a specific locale without affecting other locales.

Figure 1–1 shows that locale-specific data is loaded at runtime. In this example, French data and Japanese data are loaded into the multilingual database, but German data is not.
The locale-specific data is stored in the $ORACLE_HOME/nls/data directory. The ORA_NLS10 environment variable should be defined only when you need to change the default directory location for the locale-specific datafiles, for example when the system has multiple Oracle homes that share a single copy of the locale-specific datafiles.

A boot file is used to determine the availability of the NLS objects that can be loaded. Oracle supports both system and user boot files. The user boot file gives you the flexibility to tailor what NLS locale objects are available for the database. Also, new locale data can be added and some locale data components can be customized.

See Also: Chapter 13, "Customizing Locale"

**Architecture to Support Multilingual Applications**

The database is implemented to enable multitier applications and client/server applications to support languages for which the database is configured.

The locale-dependent operations are controlled by several parameters and environment variables on both the client and the database server. On the database server, each session started on behalf of a client may run in the same or a different
locale as other sessions, and have the same or different language requirements specified.

The database has a set of session-independent NLS parameters that are specified when the database is created. Two of the parameters specify the database character set and the national character set, an alternate Unicode character set that can be specified for NCHAR, NVARCHAR2, and NCLOB data. The parameters specify the character set that is used to store text data in the database. Other parameters, like language and territory, are used to evaluate check constraints.

If the client session and the database server specify different character sets, then the database converts character set strings automatically.

From a globalization support perspective, all applications are considered to be clients, even if they run on the same physical machine as the Oracle instance. For example, when SQL*Plus is started by the UNIX user who owns the Oracle software from the Oracle home in which the RDBMS software is installed, and SQL*Plus connects to the database through an adapter by specifying the ORACLE_SID parameter, SQL*Plus is considered a client. Its behavior is ruled by client-side NLS parameters.

Another example of an application being considered a client occurs when the middle tier is an application server. The different sessions spawned by the application server are considered to be separate client sessions.

When a client application is started, it initializes the client NLS environment from environment settings. All NLS operations performed locally are executed using these settings. Examples of local NLS operations are:

- Display formatting in Oracle Developer applications
- User OCI code that executes NLS OCI functions with OCI environment handles

When the application connects to a database, a session is created on the server. The new session initializes its NLS environment from NLS instance parameters specified in the initialization parameter file. These settings can be subsequently changed by an ALTER SESSION statement. The statement changes only the session NLS environment. It does not change the local client NLS environment. The session NLS settings are used to process SQL and PL/SQL statements that are executed on the server. For example, use an ALTER SESSION statement to set the NLS_LANGUAGE initialization parameter to Italian:

```
ALTER SESSION SET NLS_LANGUAGE=Italian;
```

Enter a SELECT statement:
SQL> SELECT last_name, hire_date, ROUND(salary/8,2) salary FROM employees;

You should see results similar to the following:

<table>
<thead>
<tr>
<th>LAST_NAME</th>
<th>HIRE_DATE</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sciarra</td>
<td>30-SET-97</td>
<td>962.5</td>
</tr>
<tr>
<td>Urman</td>
<td>07-MAR-98</td>
<td>975</td>
</tr>
<tr>
<td>Popp</td>
<td>07-DIC-99</td>
<td>862.5</td>
</tr>
</tbody>
</table>

Note that the month name abbreviations are in Italian.

Immediately after the connection has been established, if the NLS_LANG environment setting is defined on the client side, then an implicit ALTER SESSION statement synchronizes the client and session NLS environments.

See Also:
- Chapter 10, "OCI Programming in a Global Environment"
- Chapter 3, "Setting Up a Globalization Support Environment"

Using Unicode in a Multilingual Database

Unicode is a universal encoded character set that enables you to store information in any language, using a single character set. Unicode provides a unique code value for every character, regardless of the platform, program, or language.

Unicode has the following advantages:
- It simplifies character set conversion and linguistic sort functions
- It improves performance compared with native multibyte character sets
- It supports the Unicode datatype based on the Unicode standard

See Also:
- Chapter 6, "Supporting Multilingual Databases with Unicode"
- Chapter 7, "Programming with Unicode"
- "Enabling Multilingual Support with Unicode Datatypes" on page 6-8

Globalization Support Features

Oracle’s standard features include:
Language Support

The database enables you to store, process, and retrieve data in native languages. The languages that can be stored in a database are all languages written in scripts that are encoded by Oracle-supported character sets. Through the use of Unicode databases and datatypes, the Oracle database supports most contemporary languages.

Additional support is available for a subset of the languages. The database knows, for example, how to display dates using translated month names or how to sort text data according to cultural conventions.

When this manual uses the term language support, it refers to the additional language-dependent functionality (for example, displaying dates or sorting text), not to the ability to store text of a specific language.

For some of the supported languages, Oracle provides translated error messages and a translated user interface of the database utilities.

See Also:

- Chapter 3, "Setting Up a Globalization Support Environment"
- "Languages" on page A-2 for a complete list of Oracle language names and abbreviations
- "Translated Messages" on page A-4 for a list of languages into which Oracle messages are translated
Territory Support

The database supports cultural conventions that are specific to geographical locations. The default local time format, date format, and numeric and monetary conventions depend on the local territory setting. Setting different NLS parameters allows the database session to use different cultural settings. For example, you can set the euro (EUR) as the primary currency and the Japanese yen (JPY) as the secondary currency for a given database session even when the territory is defined as AMERICA.

See Also:
- Chapter 3, "Setting Up a Globalization Support Environment"
- "Territories" on page A-6 for a list of territories that are supported by the Oracle server

Date and Time Formats

Different conventions for displaying the hour, day, month, and year can be handled in local formats. For example, in the United Kingdom, the date is displayed using the DD–MON–YYYY format, while Japan commonly uses the YYYY–MM–DD format. Time zones and daylight saving support are also available.

See Also:
- Chapter 3, "Setting Up a Globalization Support Environment"
- Chapter 4, "Datetime Datatypes and Time Zone Support"
- Oracle Database SQL Reference

Monetary and Numeric Formats

Currency, credit, and debit symbols can be represented in local formats. Radix symbols and thousands separators can be defined by locales. For example, in the US, the decimal point is a dot (.), while it is a comma (,) in France. Therefore, the amount $1,234 has different meanings in different countries.

See Also: Chapter 3, "Setting Up a Globalization Support Environment"
Globalization Support Features

Calendars Feature

Many different calendar systems are in use around the world. Oracle supports seven different calendar systems: Gregorian, Japanese Imperial, ROC Official (Republic of China), Thai Buddha, Persian, English Hijrah, and Arabic Hijrah.

See Also:
- Chapter 3, "Setting Up a Globalization Support Environment"
- "Calendar Systems" on page A-28 for a list of supported calendars

Linguistic Sorting

Oracle provides linguistic definitions for culturally accurate sorting and case conversion. The basic definition treats strings as sequences of independent characters. The extended definition recognizes pairs of characters that should be treated as special cases.

Strings that are converted to upper case or lower case using the basic definition always retain their lengths. Strings converted using the extended definition may become longer or shorter.

See Also: Chapter 5, "Linguistic Sorting and String Searching"

Character Set Support

Oracle supports a large number of single-byte, multibyte, and fixed-width encoding schemes that are based on national, international, and vendor-specific standards.

See Also:
- Chapter 2, "Choosing a Character Set"
- "Character Sets" on page A-7 for a list of supported character sets

Character Semantics

Oracle provides character semantics. It is useful for defining the storage requirements for multibyte strings of varying widths in terms of characters instead of bytes.

See Also: "Length Semantics" on page 2-12
Customization of Locale and Calendar Data

You can customize locale data such as language, character set, territory, or linguistic sort using the Oracle Locale Builder.

You can customize calendars with the NLS Calendar Utility.

See Also:

- Chapter 13, "Customizing Locale"
- "Customizing Calendars with the NLS Calendar Utility" on page 13-17

Unicode Support

You can store Unicode characters in an Oracle database in two ways:

- You can create a Unicode database that enables you to store UTF-8 encoded characters as SQL CHAR datatypes.

- You can support multilingual data in specific columns by using Unicode datatypes. You can store Unicode characters into columns of the SQL NCHAR datatypes regardless of how the database character set has been defined. The NCHAR datatype is an exclusively Unicode datatype.

See Also: Chapter 6, "Supporting Multilingual Databases with Unicode"
This chapter explains how to choose a character set. It includes the following topics:

- Character Set Encoding
- Length Semantics
- Choosing an Oracle Database Character Set
- Changing the Character Set After Database Creation
- Monolingual Database Scenario
- Multilingual Database Scenarios
Character Set Encoding

When computer systems process characters, they use numeric codes instead of the graphical representation of the character. For example, when the database stores the letter A, it actually stores a numeric code that is interpreted by software as the letter. These numeric codes are especially important in a global environment because of the potential need to convert data between different character sets.

This section includes the following topics:

- What is an Encoded Character Set?
- Which Characters Are Encoded?
- What Characters Does a Character Set Support?
- How are Characters Encoded?
- Naming Convention for Oracle Character Sets

What is an Encoded Character Set?

You specify an encoded character set when you create a database. Choosing a character set determines what languages can be represented in the database. It also affects:

- How you create the database schema
- How you develop applications that process character data
- How the database works with the operating system
- Performance
- Storage required when storing character data

A group of characters (for example, alphabetic characters, ideographs, symbols, punctuation marks, and control characters) can be encoded as a character set. An encoded character set assigns unique numeric codes to each character in the character repertoire. The numeric codes are called code points or encoded values. Table 2–1 shows examples of characters that have been assigned a hexadecimal code value in the ASCII character set.

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
<th>Hexadecimal Code Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Exclamation Mark</td>
<td>21</td>
</tr>
</tbody>
</table>
The computer industry uses many encoded character sets. Character sets differ in the following ways:

- The number of characters available
- The available characters (the character repertoire)
- The scripts used for writing and the languages they represent
- The code values assigned to each character
- The encoding scheme used to represent a character

Oracle supports most national, international, and vendor-specific encoded character set standards.

See Also: "Character Sets" on page A-7 for a complete list of character sets that are supported by Oracle

### Which Characters Are Encoded?

The characters that are encoded in a character set depend on the writing systems that are represented. A writing system can be used to represent a language or group of languages. Writing systems can be classified into two categories:

- Phonetic Writing Systems
This section also includes the following topics:

- Punctuation, Control Characters, Numbers, and Symbols
- Writing Direction

**Phonetic Writing Systems**

Phonetic writing systems consist of symbols that represent different sounds associated with a language. Greek, Latin, Cyrillic, and Devanagari are all examples of phonetic writing systems based on alphabets. Note that alphabets can represent more than one language. For example, the Latin alphabet can represent many Western European languages such as French, German, and English.

Characters associated with a phonetic writing system can typically be encoded in one byte because the character repertoire is usually smaller than 256 characters.

**Ideographic Writing Systems**

Ideographic writing systems consist of ideographs or pictographs that represent the meaning of a word, not the sounds of a language. Chinese and Japanese are examples of ideographic writing systems that are based on tens of thousands of ideographs. Languages that use ideographic writing systems may also use a syllabary. Syllabaries provide a mechanism for communicating additional phonetic information. For instance, Japanese has two syllabaries: Hiragana, normally used for grammatical elements, and Katakana, normally used for foreign and onomatopoeic words.

Characters associated with an ideographic writing system typically are encoded in more than one byte because the character repertoire has tens of thousands of characters.

**Punctuation, Control Characters, Numbers, and Symbols**

In addition to encoding the script of a language, other special characters need to be encoded:

- Punctuation marks such as commas, periods, and apostrophes
- Numbers
- Special symbols such as currency symbols and math operators
- Control characters such as carriage returns and tabs
Writing Direction

Most Western languages are written left to right from the top to the bottom of the page. East Asian languages are usually written top to bottom from the right to the left of the page, although exceptions are frequently made for technical books translated from Western languages. Arabic and Hebrew are written right to left from the top to the bottom.

Numbers reverse direction in Arabic and Hebrew. Although the text is written right to left, numbers within the sentence are written left to right. For example, "I wrote 32 books" would be written as "skoob 32 etorw I". Regardless of the writing direction, Oracle stores the data in logical order. Logical order means the order that is used by someone typing a language, not how it looks on the screen.

Writing direction does not affect the encoding of a character.

What Characters Does a Character Set Support?

Different character sets support different character repertoires. Because character sets are typically based on a particular writing script, they can support more than one language. When character sets were first developed, they had a limited character repertoire. Even now there can be problems using certain characters across platforms. The following CHAR and VARCHAR characters are represented in all Oracle database character sets and can be transported to any platform:

- Uppercase and lowercase English characters A through Z and a through z
- Arabic digits 0 through 9
- The following punctuation marks: % ‘ ’ ( ) * + - . / \ ; < > = ! _ & ~ { } | ^ ? $ # @ “ ” [ ]
- The following control characters: space, horizontal tab, vertical tab, form feed

If you are using characters outside this set, then take care that your data is supported in the database character set that you have chosen.

Setting the NLS_LANG parameter properly is essential to proper data conversion. The character set that is specified by the NLS_LANG parameter should reflect the setting for the client operating system. Setting NLS_LANG correctly enables proper conversion from the client operating system character encoding to the database character set. When these settings are the same, Oracle assumes that the data being sent or received is encoded in the same character set as the database character set, so no validation or conversion is performed. This can lead to corrupt data if conversions are necessary.
During conversion from one character set to another, Oracle expects client-side data to be encoded in the character set specified by the NLS_LANG parameter. If you put other values into the string (for example, by using the CHR or CONVERT SQL functions), then the values may be corrupted when they are sent to the database because they are not converted properly. If you have configured the environment correctly and if the database character set supports the entire repertoire of character data that may be input into the database, then you do not need to change the current database character set. However, if your enterprise becomes more global and you have additional characters or new languages to support, then you may need to choose a character set with a greater character repertoire. Oracle Corporation recommends that you use Unicode databases and datatypes in these cases.

See Also:
- Chapter 6, "Supporting Multilingual Databases with Unicode"
- Oracle Database SQL Reference for more information about the CHR and CONVERT SQL functions
- "Displaying a Code Chart with the Oracle Locale Builder" on page 13-18

ASCII Encoding
Table 2–2 shows how the ASCII character is encoded. Row and column headings denote hexadecimal digits. To find the encoded value of a character, read the column number followed by the row number. For example, the code value of the character A is 0x41.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>DLE</td>
<td>SP</td>
<td>0</td>
<td>@</td>
<td>P</td>
<td>'</td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
<td>!</td>
<td>1</td>
<td>A</td>
<td>Q</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>DC2</td>
<td>&quot;</td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>NAK</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>SYN</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>ETB</td>
<td>'</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>g</td>
</tr>
</tbody>
</table>
Character sets have evolved to meet the needs of users around the world. New character sets have been created to support languages besides English. Typically, these new character sets support a group of related languages based on the same script. For example, the ISO 8859 character set series was created to support different European languages. Table 2-3 shows the languages that are supported by the ISO 8859 character sets.

### Table 2-2 7-Bit ASCII Character Set (Cont.)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>BS</td>
<td>CAN</td>
<td>(</td>
<td>8</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>TAB</td>
<td>EM</td>
<td>)</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>y</td>
</tr>
<tr>
<td>A</td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>:</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
</tr>
<tr>
<td>B</td>
<td>VT</td>
<td>ESC</td>
<td>+</td>
<td>;</td>
<td>K</td>
<td>[</td>
<td>k</td>
<td>]</td>
</tr>
<tr>
<td>C</td>
<td>FF</td>
<td>FS</td>
<td>,</td>
<td>&lt;</td>
<td>L</td>
<td>\</td>
<td>l</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>CR</td>
<td>GS</td>
<td>-</td>
<td>=</td>
<td>M</td>
<td>}</td>
<td>m</td>
<td>}</td>
</tr>
<tr>
<td>E</td>
<td>SO</td>
<td>RS</td>
<td>.</td>
<td>&gt;</td>
<td>N</td>
<td>^</td>
<td>n</td>
<td>~</td>
</tr>
<tr>
<td>F</td>
<td>SI</td>
<td>US</td>
<td>/</td>
<td>?</td>
<td>O</td>
<td>_</td>
<td>o</td>
<td>DEL</td>
</tr>
</tbody>
</table>

Character sets have evolved to meet the needs of users around the world. New character sets have been created to support languages besides English. Typically, these new character sets support a group of related languages based on the same script. For example, the ISO 8859 character set series was created to support different European languages. Table 2-3 shows the languages that are supported by the ISO 8859 character sets.
Character sets evolved and provided restricted multilingual support. They were restricted in the sense that they were limited to groups of languages based on similar scripts. More recently, universal character sets have been regarded as a more useful solution to multilingual support. Unicode is one such universal character set that encompasses most major scripts of the modern world. The Unicode character set supports more than 94,000 characters.
How are Characters_encoded?

Different types of encoding schemes have been created by the computer industry. The character set you choose affects what kind of encoding scheme is used. This is important because different encoding schemes have different performance characteristics. These characteristics can influence your database schema and application development. The character set you choose uses one of the following types of encoding schemes:

- Single-Byte Encoding Schemes
- Multibyte Encoding Schemes

Single-Byte Encoding Schemes

Single-byte encoding schemes are efficient. They take up the least amount of space to represent characters and are easy to process and program with because one character can be represented in one byte. Single-byte encoding schemes are classified as one of the following:

- 7-bit encoding schemes
  
  Single-byte 7-bit encoding schemes can define up to 128 characters and normally support just one language. One of the most common single-byte character sets, used since the early days of computing, is ASCII (American Standard Code for Information Interchange).

- 8-bit encoding schemes
  
  Single-byte 8-bit encoding schemes can define up to 256 characters and often support a group of related languages. One example is ISO 8859-1, which supports many Western European languages. Figure 2–1 shows the ISO 8859-1 8-bit encoding scheme.

See Also: Chapter 6, “Supporting Multilingual Databases with Unicode”
Multibyte Encoding Schemes

Multibyte encoding schemes are needed to support ideographic scripts used in Asian languages like Chinese or Japanese because these languages use thousands of characters. These encoding schemes use either a fixed number or a variable number of bytes to represent each character.

- Fixed-width multibyte encoding schemes

  In a fixed-width multibyte encoding scheme, each character is represented by a fixed number of bytes. The number of bytes is at least two in a multibyte encoding scheme.

- Variable-width multibyte encoding schemes

  A variable-width encoding scheme uses one or more bytes to represent a single character. Some multibyte encoding schemes use certain bits to indicate the number of bytes that represents a character. For example, if two bytes is the maximum number of bytes used to represent a character, then the most significant bit can be used to indicate whether that byte is a single-byte character or the first byte of a double-byte character.

- Shift-sensitive variable-width multibyte encoding schemes
Some variable-width encoding schemes use control codes to differentiate between single-byte and multibyte characters with the same code values. A shift-out code indicates that the following character is multibyte. A shift-in code indicates that the following character is single-byte. Shift-sensitive encoding schemes are used primarily on IBM platforms. Note that ISO-2022 character sets cannot be used as database character sets, but they can be used for applications such as a mail server.

### Naming Convention for Oracle Character Sets

Oracle uses the following naming convention for Oracle character set names:

\[
\text{<region><number of bits used to represent a character><standard character set name>[S|C]\n}\]

The parts of the names in angle brackets are concatenated. The optional S or C is used to differentiate character sets that can be used only on the server (S) or only on the client (C).

---

**Note:** Use the server character set (S) on the Macintosh platform. The Macintosh client character sets are obsolete. On EBCDIC platforms, use the server character set (S) on the server and the client character set (C) on the client.

---

**Note:** UTF8 and UTFE are exceptions to the naming convention.

---

Table 2–4 shows examples of Oracle character set names.

**Table 2–4  Examples of Oracle Character Set Names**

<table>
<thead>
<tr>
<th>Oracle Character Set Name</th>
<th>Description</th>
<th>Region</th>
<th>Number of Bits Used to Represent a Character</th>
<th>Standard Character Set Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>US7ASCII</td>
<td>U.S. 7-bit ASCII</td>
<td>US</td>
<td>7</td>
<td>ASCII</td>
</tr>
<tr>
<td>WE8ISO8859P1</td>
<td>Western European 8-bit ISO 8859 Part 1</td>
<td>WE (Western Europe)</td>
<td>8</td>
<td>ISO8859 Part 1</td>
</tr>
</tbody>
</table>
Length Semantics

In single-byte character sets, the number of bytes and the number of characters in a string are the same. In multibyte character sets, a character or code point consists of one or more bytes. Calculating the number of characters based on byte lengths can be difficult in a variable-width character set. Calculating column lengths in bytes is called byte semantics, while measuring column lengths in characters is called character semantics.

Character semantics were introduced in Oracle9i. Character semantics is useful for defining the storage requirements for multibyte strings of varying widths. For example, in a Unicode database (AL32UTF8), suppose that you need to define a VARCHAR2 column that can store up to five Chinese characters together with five English characters. Using byte semantics, this column requires 15 bytes for the Chinese characters, which are three bytes long, and 5 bytes for the English characters, which are one byte long, for a total of 20 bytes. Using character semantics, the column requires 10 characters.

The following expressions use byte semantics:

- VARCHAR2(20 BYTE)
- SUBSTRB(string, 1, 20)

Note the BYTE qualifier in the VARCHAR2 expression and the B suffix in the SQL function name.

The following expressions use character semantics:

- VARCHAR2(10 CHAR)
- SUBSTR(string, 1, 10)

Note the CHAR qualifier in the VARCHAR2 expression.

Table 2–4  Examples of Oracle Character Set Names (Cont.)

<table>
<thead>
<tr>
<th>Oracle Character Set Name</th>
<th>Description</th>
<th>Region</th>
<th>Number of Bits Used to Represent a Character</th>
<th>Standard Character Set Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA16SJIS</td>
<td>Japanese 16-bit Shifted Japanese Industrial Standard</td>
<td>JA</td>
<td>16</td>
<td>SJIS</td>
</tr>
</tbody>
</table>

Length Semantics

<table>
<thead>
<tr>
<th>Oracle Character Set Name</th>
<th>Description</th>
<th>Region</th>
<th>Number of Bits Used to Represent a Character</th>
<th>Standard Character Set Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA16SJIS</td>
<td>Japanese 16-bit Shifted Japanese Industrial Standard</td>
<td>JA</td>
<td>16</td>
<td>SJIS</td>
</tr>
</tbody>
</table>
The NLS_LENGTH_SEMANTICS initialization parameter determines whether a new column of character datatype uses byte or character semantics. The default value of the parameter is BYTE. The BYTE and CHAR qualifiers shown in the VARCHAR2 definitions should be avoided when possible because they lead to mixed-semantics databases. Instead, set NLS_LENGTH_SEMANTICS in the initialization parameter file and define column datatypes to use the default semantics based on the value of NLS_LENGTH_SEMANTICS.

Byte semantics is the default for the database character set. Character length semantics is the default and the only allowable kind of length semantics for NCHAR datatypes. The user cannot specify the CHAR or BYTE qualifier for NCHAR definitions.

Consider the following example:

```sql
CREATE TABLE employees
( employee_id NUMBER(4)
 , last_name NVARCHAR2(10)
 , job_id NVARCHAR2(9)
 , manager_id NUMBER(4)
 , hire_date DATE
 , salary NUMBER(7,2)
 , department_id NUMBER(2)
 ) ;
```

When the NCHAR character set is AL16UTF16, last_name can hold up to 10 Unicode code points. When the NCHAR character set is AL16UTF16, last_name can hold up to 20 bytes.

**Figure 2–2** shows the number of bytes needed to store different kinds of characters in the UTF-8 character set. The ASCII characters requires one byte, the Latin and Greek characters require two bytes, the Asian character requires three bytes, and the supplementary character requires four bytes of storage.
Choosing an Oracle Database Character Set

Oracle uses the database character set for:

- Data stored in SQL CHAR datatypes (CHAR, VARCHAR2, CLOB, and LONG)
- Identifiers such as table names, column names, and PL/SQL variables
- Entering and storing SQL and PL/SQL source code

See Also:

- "SQL Functions for Different Length Semantics" on page 9-6 for more information about the SUBSTR and SUBSTRB functions
- "Length Semantics" on page 3-44 for more information about the NLS_LENGTH_SEMANTICS initialization parameter
- Chapter 6, "Supporting Multilingual Databases with Unicode" for more information about Unicode and the NCHAR datatype
- Oracle Database SQL Reference for more information about the SUBSTRB and SUBSTR functions and the BYTE and CHAR qualifiers for character datatypes

Choosing an Oracle Database Character Set

Oracle uses the database character set for:

- Data stored in SQL CHAR datatypes (CHAR, VARCHAR2, CLOB, and LONG)
- Identifiers such as table names, column names, and PL/SQL variables
- Entering and storing SQL and PL/SQL source code

Figure 2–2  Bytes of Storage for Different Kinds of Characters

![Diagram showing byte storage for different kinds of characters]
Choosing an Oracle Database Character Set

The character encoding scheme used by the database is defined as part of the
CREATE DATABASE statement. All SQL CHAR datatype columns (CHAR, CLOB,
VARCHAR2, and LONG), including columns in the data dictionary, have their data
stored in the database character set. In addition, the choice of database character set
determines which characters can name objects in the database. SQL NCHAR datatype
columns (NCHAR, NCLOB, and NVARCHAR2) use the national character set.

Note: CLOB data is stored in a format that is compatible with
UCS-2 if the database character set is multibyte. If the database
character set is single-byte, then CLOB data is stored in the
database character set.

After the database is created, you cannot change the character sets, with some
exceptions, without re-creating the database.

Consider the following questions when you choose an Oracle character set for the
database:

- What languages does the database need to support now?
- What languages does the database need to support in the future?
- Is the character set available on the operating system?
- What character sets are used on clients?
- How well does the application handle the character set?
- What are the performance implications of the character set?
- What are the restrictions associated with the character set?

The Oracle character sets are listed in "Character Sets" on page A-7. They are named
according to the languages and regions in which they are used. Some character sets
that are named for a region are also listed explicitly by language.

If you want to see the characters that are included in a character set, then:

- Check national, international, or vendor product documentation or standards
documents
- Use Oracle Locale Builder

This section contains the following topics:

- Current and Future Language Requirements
Choosing an Oracle Database Character Set

- Client Operating System and Application Compatibility
- Character Set Conversion Between Clients and the Server
- Performance Implications of Choosing a Database Character Set
- Restrictions on Database Character Sets
- Choosing a National Character Set
- Summary of Supported Datatypes

See Also:

- "UCS-2 Encoding" on page 6-4
- "Choosing a National Character Set" on page 2-19
- "Changing the Character Set After Database Creation" on page 2-20
- Appendix A, "Locale Data"
- Chapter 13, "Customizing Locale"

Current and Future Language Requirements

Several character sets may meet your current language requirements. Consider future language requirements when you choose a database character set. If you expect to support additional languages in the future, then choose a character set that supports those languages to prevent the need to migrate to a different character set later.

Client Operating System and Application Compatibility

The database character set is independent of the operating system because Oracle has its own globalization architecture. For example, on an English Windows operating system, you can create and run a database with a Japanese character set. However, when an application in the client operating system accesses the database, the client operating system must be able to support the database character set with appropriate fonts and input methods. For example, you cannot insert or retrieve Japanese data on the English Windows operating system without first installing a Japanese font and input method. Another way to insert and retrieve Japanese data is to use a Japanese operating system remotely to access the database server.
Character Set Conversion Between Clients and the Server

If you choose a database character set that is different from the character set on the client operating system, then the Oracle database can convert the operating system character set to the database character set. Character set conversion has the following disadvantages:

- Potential data loss
- Increased overhead

Character set conversions can sometimes cause data loss. For example, if you are converting from character set A to character set B, then the destination character set B must have the same character set repertoire as A. Any characters that are not available in character set B are converted to a replacement character. The replacement character is often specified as a question mark or as a linguistically related character. For example, ä (a with an umlaut) may be converted to a. If you have distributed environments, then consider using character sets with similar character repertoires to avoid loss of data.

Character set conversion may require copying strings between buffers several times before the data reaches the client. The database character set should always be a superset or equivalent of the native character set of the client’s operating system. The character sets used by client applications that access the database usually determine which superset is the best choice.

If all client applications use the same character set, then that character set is usually the best choice for the database character set. When client applications use different character sets, the database character set should be a superset of all the client character sets. This ensures that every character is represented when converting from a client character set to the database character set.

See Also: Chapter 11, "Character Set Migration"

Performance Implications of Choosing a Database Character Set

For best performance, choose a character set that avoids character set conversion and uses the most efficient encoding for the languages desired. Single-byte character sets result in better performance than multibyte character sets, and they also are the most efficient in terms of space requirements. However, single-byte character sets limit how many languages you can support.
Restrictions on Database Character Sets

ASCII-based character sets are supported only on ASCII-based platforms. Similarly, you can use an EBCDIC-based character set only on EBCDIC-based platforms.

The database character set is used to identify SQL and PL/SQL source code. In order to do this, it must have either EBCDIC or 7-bit ASCII as a subset, whichever is native to the platform. Therefore, it is not possible to use a fixed-width, multibyte character set as the database character set. Currently, only the AL16UTF16 character set cannot be used as a database character set.

Restrictions on Character Sets Used to Express Names

Table 2–5 lists the restrictions on the character sets that can be used to express names.

**Table 2–5  Restrictions on Character Sets Used to Express Names**

<table>
<thead>
<tr>
<th>Name</th>
<th>Single-Byte</th>
<th>Variable Width</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>column names</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>schema objects</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>comments</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>database link names</td>
<td>Yes</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>database names</td>
<td>Yes</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>file names (datafile, log file, control file, initialization parameter file)</td>
<td>Yes</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>instance names</td>
<td>Yes</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>directory names</td>
<td>Yes</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>keywords</td>
<td>Yes</td>
<td>No</td>
<td>Can be expressed in English ASCII or EBCDIC characters only</td>
</tr>
<tr>
<td>Recovery Manager file names</td>
<td>Yes</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>rollback segment names</td>
<td>Yes</td>
<td>No</td>
<td>The ROLLBACK_SEGMENT parameter does not support NLS</td>
</tr>
<tr>
<td>stored script names</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>tablespace names</td>
<td>Yes</td>
<td>No</td>
<td>-</td>
</tr>
</tbody>
</table>
For a list of supported string formats and character sets, including LOB data (LOB, BLOB, CLOB, and NCLOB), see Table 2–7 on page 2-20.

**Choosing a National Character Set**

A **national character set** is an alternate character set that enables you to store Unicode character data in a database that does not have a Unicode database character set. Other reasons for choosing a national character set are:

- The properties of a different character encoding scheme may be more desirable for extensive character processing operations
- Programming in the national character set is easier

SQL NCHAR, NVARCHAR2, and NCLOB datatypes have been redefined to support Unicode data only. You can use either the UTF8 or the AL16UTF16 character set. The default is AL16UTF16.

**See Also:** Chapter 6, "Supporting Multilingual Databases with Unicode"

**Summary of Supported Datatypes**

Table 2–6 lists the datatypes that are supported for different encoding schemes.

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Single Byte</th>
<th>Multibyte Non-Unicode</th>
<th>Multibyte Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>VARCHAR2</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>NCHAR</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>NVARCHAR2</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>BLOB</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CLOB</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LONG</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>NCLOB</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Note:** BLOBs process characters as a series of byte sequences. The data is not subject to any NLS-sensitive operations.
Changing the Character Set After Database Creation

Table 2–7 lists the SQL datatypes that are supported for abstract datatypes.

<table>
<thead>
<tr>
<th>Abstract Datatype</th>
<th>CHAR</th>
<th>NCHAR</th>
<th>BLOB</th>
<th>CLOB</th>
<th>NCLOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Collection</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

You can create an abstract datatype with the `NCHAR` attribute as follows:

```
SQL> CREATE TYPE t1 AS OBJECT (a NCHAR(10));
Type created.
SQL> CREATE TABLE t1 (a t1);
Table created.
```

See Also: Oracle Database Application Developer’s Guide - Object-Relational Features for more information about objects and collections

Changing the Character Set After Database Creation

You may wish to change the database character set after the database has been created. For example, you may find that the number of languages that need to be supported in your database has increased. In most cases, you need to do a full export/import to properly convert all data to the new character set. However, if, and only if, the new character set is a strict superset of the current character set, then it is possible to use the `ALTER DATABASE CHARACTER SET` statement to expedite the change in the database character set.

See Also:
- Chapter 11, "Character Set Migration"
- Oracle Database Upgrade Guide for more information about exporting and importing data
- Oracle Database SQL Reference for more information about the ALTER DATABASE CHARACTER SET statement

Monolingual Database Scenario

The simplest example of a database configuration is a client and a server that run in the same language environment and use the same character set. This monolingual
scenario has the advantage of fast response because the overhead associated with character set conversion is avoided. Figure 2–3 shows a database server and a client that use the same character set. The Japanese client and the server both use the JA16EUC character set.

**Figure 2–3  Monolingual Database Scenario**

![Diagram of a monolingual database scenario](image)

You can also use a multitier architecture. Figure 2–4 shows an application server between the database server and the client. The application server and the database server use the same character set in a monolingual scenario. The server, the application server, and the client use the JA16EUC character set.

**Figure 2–4  Multitier Monolingual Database Scenario**

![Diagram of a multitier monolingual database scenario](image)

**Character Set Conversion in a Monolingual Scenario**

Character set conversion may be required in a client/server environment if a client application resides on a different platform than the server and if the platforms do not use the same character encoding schemes. Character data passed between client and server must be converted between the two encoding schemes. Character conversion occurs automatically and transparently through Oracle Net.
You can convert between any two character sets. Figure 2–5 shows a server and one client with the JA16EUC Japanese character set. The other client uses the JA16SJIS Japanese character set.

Figure 2–5  Character Set Conversion

When a target character set does not contain all of the characters in the source data, replacement characters are used. If, for example, a server uses US7ASCII and a German client uses WE8ISO8859P1, then the German character ß is replaced with ? and ä is replaced with a.

Replacement characters may be defined for specific characters as part of a character set definition. When a specific replacement character is not defined, a default replacement character is used. To avoid the use of replacement characters when converting from a client character set to a database character set, the server character set should be a superset of all the client character sets.

Figure 2–6 shows that data loss occurs when the database character set does not include all of the characters in the client character set. The database character set is US7ASCII. The client’s character set is WE8MSWIN1252, and the language used by the client is German. When the client inserts a string that contains ß, the database replaces ß with ?, resulting in lost data.
If German data is expected to be stored on the server, then a database character set that supports German characters should be used for both the server and the client to avoid data loss and conversion overhead.

When one of the character sets is a variable-width multibyte character set, conversion can introduce noticeable overhead. Carefully evaluate your situation and choose character sets to avoid conversion as much as possible.

**Multilingual Database Scenarios**

Multilingual support can be restricted or unrestricted. This section contains the following topics:

- Restricted Multilingual Support
- Unrestricted Multilingual Support
Restricted Multilingual Support

Some character sets support multiple languages because they have related writing systems or scripts. For example, the WE8ISO8859P1 Oracle character set supports the following Western European languages:

- Catalan
- Danish
- Dutch
- English
- Finnish
- French
- German
- Icelandic
- Italian
- Norwegian
- Portuguese
- Spanish
- Swedish

These languages all use a Latin-based writing script.

When you use a character set that supports a group of languages, your database has restricted multilingual support.

Figure 2–7 shows a Western European server that used the WE8ISO8850P1 Oracle character set, a French client that uses the same character set as the server, and a German client that uses the WE8DEC character set. The German client requires character conversion because it is using a different character set than the server.
Unrestricted Multilingual Support

If you need unrestricted multilingual support, then use a universal character set such as Unicode for the server database character set. Unicode has two major encoding schemes:

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

The database provides support for UTF-8 as a database character set and both UTF-8 and UTF-16 as national character sets.

Character set conversion between a UTF-8 database and any single-byte character set introduces very little overhead.

Conversion between UTF-8 and any multibyte character set has some overhead. There is no data loss from conversion with the following exceptions:

- Some multibyte character sets do not support user-defined characters during character set conversion to and from UTF-8.
- Some Unicode characters are mapped to more than one character in another character set. For example, one Unicode character is mapped to three characters.
in the JA16SJIS character set. This means that a round-trip conversion may not result in the original JA16SJIS character.

Figure 2–8 shows a server that uses the AL32UTF8 Oracle character set that is based on the Unicode UTF-8 character set.

Figure 2–8 Unrestricted Multilingual Support Scenario in a Client/Server Configuration

There are four clients:
- A French client that uses the WE8ISO8859P1 Oracle character set
- A German client that uses the WE8DEC character set
- A Japanese client that uses the JA16EUC character set
- A Japanese client that uses the JA16SJIS character set
A Japanese client that used the JA16SJIS character set

Character conversion takes place between each client and the server, but there is no data loss because AL32UTF8 is a universal character set. If the German client tries to retrieve data from one of the Japanese clients, then all of the Japanese characters in the data are lost during the character set conversion.

Figure 2–9 shows a Unicode solution for a multitier configuration.

Figure 2–9  Multitier Unrestricted Multilingual Support Scenario in a Multitier Configuration

The database, the application server, and each client use the AL32UTF8 character set. This eliminates the need for character conversion even though the clients are French, German, and Japanese.

See Also:  Chapter 6, “Supporting Multilingual Databases with Unicode”
This chapter tells how to set up a globalization support environment. It includes the following topics:

- Setting NLS Parameters
- Choosing a Locale with the NLS_LANG Environment Variable
- NLS Database Parameters
- Language and Territory Parameters
- Date and Time Parameters
- Calendar Definitions
- Numeric and List Parameters
- Monetary Parameters
- Linguistic Sort Parameters
- Character Set Conversion Parameter
- Length Semantics
Setting NLS Parameters

NLS parameters determine the locale-specific behavior on both the client and the server. NLS parameters can be specified in the following ways:

- As initialization parameters on the server
  You can include parameters in the initialization parameter file to specify a default session NLS environment. These settings have no effect on the client side; they control only the server’s behavior. For example:
  ```
  NLS_TERRITORY = "CZECH REPUBLIC"
  ```

- As environment variables on the client
  You can use NLS environment variables, which may be platform-dependent, to specify locale-dependent behavior for the client and also to override the default values set for the session in the initialization parameter file. For example, on a UNIX system:
  ```
  % setenv NLS_SORT FRENCH
  ```

- With the `ALTER SESSION` statement
  NLS parameters that are set in an `ALTER SESSION` statement can be used to override the default values that are set for the session in the initialization parameter file or set by the client with environment variables.
  ```
  ALTER SESSION SET NLS_SORT = FRENCH;
  ```

  **See Also:** Oracle Database SQL Reference for more information about the `ALTER SESSION` statement

- In SQL functions
  NLS parameters can be used explicitly to hardcode NLS behavior within a SQL function. This practice overrides the default values that are set for the session in the initialization parameter file, set for the client with environment variables, or set for the session by the `ALTER SESSION` statement. For example:
  ```
  TO_CHAR(hiredate, 'DD/MON/YYYY', 'nls_date_language = FRENCH')
  ```

  **See Also:** Oracle Database SQL Reference for more information about SQL functions, including the `TO_CHAR` function
Table 3–1 shows the precedence order of the different methods of setting NLS parameters. Higher priority settings override lower priority settings. For example, a default value has the lowest priority and can be overridden by any other method.

Table 3–1 Methods of Setting NLS Parameters and Their Priorities

<table>
<thead>
<tr>
<th>Priority</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (highest)</td>
<td>Explicitly set in SQL functions</td>
</tr>
<tr>
<td>2</td>
<td>Set by an ALTER SESSION statement</td>
</tr>
<tr>
<td>3</td>
<td>Set as an environment variable</td>
</tr>
<tr>
<td>4</td>
<td>Specified in the initialization parameter file</td>
</tr>
<tr>
<td>5</td>
<td>Default</td>
</tr>
</tbody>
</table>

Table 3–2 lists the available NLS parameters. Because the SQL function NLS parameters can be specified only with specific functions, the table does not show the SQL function scope.

Table 3–2 NLS Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Scope:</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLSCALENDAR</td>
<td>Calendar system</td>
<td>Gregorian</td>
<td>I, E, A</td>
</tr>
<tr>
<td>NLS_COMP</td>
<td>SQL, PL/SQL operator comparison</td>
<td>BINARY</td>
<td>I, E, A</td>
</tr>
<tr>
<td>NLS_CREDIT</td>
<td>Credit accounting symbol</td>
<td>Derived from NLS_TERRITORY</td>
<td>E</td>
</tr>
<tr>
<td>NLS_CURRENCY</td>
<td>Local currency symbol</td>
<td>Derived from NLS_TERRITORY</td>
<td>I, E, A</td>
</tr>
<tr>
<td>NLS_DATE_FORMAT</td>
<td>Date format</td>
<td>Derived from NLS_TERRITORY</td>
<td>I, E, A</td>
</tr>
<tr>
<td>NLS_DATE_LANGUAGE</td>
<td>Language for day and month names</td>
<td>Derived from NLS_LANGUAGE</td>
<td>I, E, A</td>
</tr>
<tr>
<td>NLS_DEBIT</td>
<td>Debit accounting symbol</td>
<td>Derived from NLS_TERRITORY</td>
<td>E</td>
</tr>
</tbody>
</table>
### Table 3–2  NLS Parameters (Cont.)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Scope:</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLS_ISO_CURRENCY</td>
<td>ISO international currency symbol</td>
<td>Derived from NLS_TERRITORY</td>
<td>I, E, A</td>
</tr>
<tr>
<td>NLS_LANG</td>
<td>Language, territory, character set</td>
<td>AMERICAN_AMERICA.US7ASCII</td>
<td>E</td>
</tr>
<tr>
<td><strong>See Also:</strong> &quot;Choosing a Locale with the NLS_LANG Environment Variable” on page 3-5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NLS_LANGUAGE</td>
<td>Language</td>
<td>Derived from NLS_LANG</td>
<td>I, A</td>
</tr>
<tr>
<td>NLS_LENGTH_SEMANTICS</td>
<td>How strings are treated</td>
<td>BYTE</td>
<td>I, A</td>
</tr>
<tr>
<td>NLS_LIST_SEPARATOR</td>
<td>Character that separates items in a list</td>
<td>Derived from NLS_TERRITORY</td>
<td>E</td>
</tr>
<tr>
<td>NLS_MONETARY_CHARACTERS</td>
<td>Monetary symbol for dollar and cents (or their equivalents)</td>
<td>Derived from NLS_TERRITORY</td>
<td>E</td>
</tr>
<tr>
<td>NLS_NCHAR_CONV_EXCP</td>
<td>Reports data loss during a character type conversion</td>
<td>FALSE</td>
<td>I, A</td>
</tr>
<tr>
<td>NLS_NUMERIC_CHARACTERS</td>
<td>Decimal character and group separator</td>
<td>Derived from NLS_TERRITORY</td>
<td>I, E, A</td>
</tr>
<tr>
<td>NLS_SORT</td>
<td>Character sort sequence</td>
<td>Derived from NLS_LANGUAGE</td>
<td>I, E, A</td>
</tr>
<tr>
<td>NLS_TERRITORY</td>
<td>Territory</td>
<td>Derived from NLS_LANG</td>
<td>I, A</td>
</tr>
<tr>
<td>NLS_TIMESTAMP_FORMAT</td>
<td>Timestamp</td>
<td>Derived from NLS_TERRITORY</td>
<td>I, E, A</td>
</tr>
<tr>
<td>NLS_TIMESTAMP_TZ_FORMAT</td>
<td>Timestamp with time zone</td>
<td>Derived from NLS_TERRITORY</td>
<td>I, E, A</td>
</tr>
<tr>
<td>NLS_DUAL_CURRENCY</td>
<td>Dual currency symbol</td>
<td>Derived from NLS_TERRITORY</td>
<td>I, E, A</td>
</tr>
</tbody>
</table>
Choosing a Locale with the NLS_LANG Environment Variable

A locale is a linguistic and cultural environment in which a system or program is running. Setting the NLS_LANG environment parameter is the simplest way to specify locale behavior for Oracle software. It sets the language and territory used by the client application and the database server. It also sets the client’s character set, which is the character set for data entered or displayed by a client program.

NLS_LANG is set as a local environment variable on UNIX platforms. NLS_LANG is set in the registry on Windows platforms.

The NLS_LANG parameter has three components: language, territory, and character set. Specify it in the following format, including the punctuation:

```
NLS_LANG = language_territory.charset
```

For example, if the Oracle Installer does not populate NLS_LANG, then its value by default is AMERICAN_AMERICA.US7ASCII. The language is AMERICAN, the territory is AMERICA, and the character set is US7ASCII.

Each component of the NLS_LANG parameter controls the operation of a subset of globalization support features:

- **language**
  
  Specifies conventions such as the language used for Oracle messages, sorting, day names, and month names. Each supported language has a unique name; for example, AMERICAN, FRENCH, or GERMAN. The language argument specifies default values for the territory and character set arguments. If the language is not specified, then the value defaults to AMERICAN.

- **territory**
  
  Specifies conventions such as the default date, monetary, and numeric formats. Each supported territory has a unique name; for example, AMERICA, FRANCE, or CANADA. If the territory is not specified, then the value is derived from the language value.

- **charset**
  
  Specifies the character set used by the client application (normally the Oracle character set that corresponds to the user’s terminal character set or the OS character set). Each supported character set has a unique acronym, for example, US7ASCII, WE8ISO8859P1, WE8DEC, WE8MSWIN1252, or JA16EUC. Each language has a default character set associated with it.
Choosing a Locale with the NLS_LANG Environment Variable

---

**Note:** All components of the `NLS_LANG` definition are optional; any item that is not specified uses its default value. If you specify territory or character set, then you *must* include the preceding delimiter [underscore (_) for territory, period (.) for character set]. Otherwise, the value is parsed as a language name.

For example, to set only the territory portion of `NLS_LANG`, use the following format: `NLS_LANG=_JAPAN`

The three components of `NLS_LANG` can be specified in many combinations, as in the following examples:

```
NLS_LANG = AMERICAN_AMERICA.WE8MSWIN1252
NLS_LANG = FRENCH_CANADA.WE8ISO8859P1
NLS_LANG = JAPANESE_JAPAN.JA16EUC
```

Note that illogical combinations can be set but do not work properly. For example, the following specification tries to support Japanese by using a Western European character set:

```
NLS_LANG = JAPANESE_JAPAN.WE8ISO8859P1
```

Because the `WE8ISO8859P1` character set does not support any Japanese characters, you cannot store or display Japanese data if you use this definition for `NLS_LANG`.

The rest of this section includes the following topics:

- Specifying the Value of `NLS_LANG`
- Overriding Language and Territory Specifications
- Locale Variants

See Also:

- Appendix A, "Locale Data" for a complete list of supported languages, territories, and character sets
- Your operating system documentation for information about additional globalization settings that may be necessary for your platform
Choosing a Locale with the NLS_LANG Environment Variable

Specifying the Value of NLS_LANG

Set NLS_LANG as an environment variable. For example, in a UNIX operating system C-shell session, you can specify the value of NLS_LANG by entering a statement similar to the following:

```
% setenv NLS_LANG FRENCH_FRANCE.WE8ISO8859P1
```

Because NLS_LANG is an environment variable, it is read by the client application at startup time. The client communicates the information defined by NLS_LANG to the server when it connects to the database server.

The following examples show how date and number formats are affected by the NLS_LANG parameter.

**Example 3–1 Setting NLS_LANG to American_America.WE8ISO8859P1**

Set NLS_LANG so that the language is AMERICAN, the territory is AMERICA, and the Oracle character set is WE8ISO8859P1:

```
% setenv NLS_LANG American_America.WE8ISO8859P1
```

Enter a SELECT statement:

```
SQL> SELECT last_name, hire_date, ROUND(salary/8,2) salary FROM employees;
```

You should see results similar to the following:

<table>
<thead>
<tr>
<th>LAST_NAME</th>
<th>HIRE_DATE</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sciarra</td>
<td>30-SEP-97</td>
<td>962.5</td>
</tr>
<tr>
<td>Urman</td>
<td>07-MAR-98</td>
<td>975</td>
</tr>
<tr>
<td>Popp</td>
<td>07-DEC-99</td>
<td>862.5</td>
</tr>
</tbody>
</table>

**Example 3–2 Setting NLS_LANG to French_France.WE8ISO8859P1**

Set NLS_LANG so that the language is FRENCH, the territory is FRANCE, and the Oracle character set is WE8ISO8859P1:

```
% setenv NLS_LANG French_France.WE8ISO8859P1
```

Then the query shown in Example 3–1 returns the following output:

<table>
<thead>
<tr>
<th>LAST_NAME</th>
<th>HIRE_DATE</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sciarra</td>
<td>30/09/97</td>
<td>962,5</td>
</tr>
<tr>
<td>Urman</td>
<td>07/03/98</td>
<td>975</td>
</tr>
</tbody>
</table>
Choosing a Locale with the NLS_LANG Environment Variable

Note that the date format and the number format have changed. The numbers have not changed, because the underlying data is the same.

Overriding Language and Territory Specifications

The NLS_LANG parameter sets the language and territory environment used by both the server session (for example, SQL command execution) and the client application (for example, display formatting in Oracle tools). Using this parameter ensures that the language environments of both the database and the client application are automatically the same.

The language and territory components of the NLS_LANG parameter determine the default values for other detailed NLS parameters, such as date format, numeric characters, and linguistic sorting. Each of these detailed parameters can be set in the client environment to override the default values if the NLS_LANG parameter has already been set.

If the NLS_LANG parameter is not set, then the server session environment remains initialized with values of NLS_LANGUAGE, NLS_TERRITORY, and other NLS instance parameters from the initialization parameter file. You can modify these parameters and restart the instance to change the defaults.

You might want to modify the NLS environment dynamically during the session. To do so, you can use the ALTER SESSION statement to change NLS_LANGUAGE, NLS_TERRITORY, and other NLS parameters.

Note: You cannot modify the setting for the client character set with the ALTER SESSION statement.

The ALTER SESSION statement modifies only the session environment. The local client NLS environment is not modified, unless the client explicitly retrieves the new settings and modifies its local environment.

See Also:

- "Overriding Default Values for NLS_LANGUAGE and NLS_TERRITORY During a Session" on page 3-18
- Oracle Database SQL Reference
Locale Variants

Before Oracle Database 10g Release 1 (10.1), Oracle defined language and territory definitions separately. This resulted in the definition of a territory being independent of the language setting of the user. In Oracle Database 10g Release 1 (10.1), some territories can have different date, time, number, and monetary formats based on the language setting of a user. This type of language-dependent territory definition is called a locale variant.

For the variant to work properly, both NLS_TERRITORY and NLS_LANGUAGE must be specified. For example, if NLS_LANGUAGE is specified as DUTCH and NLS_TERRITORY is not set, then the territory behavior is THE NETHERLANDS. If NLS_TERRITORY is set to BELGIUM and NLS_LANGUAGE is not set or it is set to DUTCH, then date, time, number, and monetary formats are based on DUTCH behavior. By contrast, if NLS_TERRITORY is set to BELGIUM and NLS_LANGUAGE is set to FRENCH, then date, time, number, and monetary formats are based on FRENCH behavior.

Table 3–3 shows the territories that have been enhanced to support variations. Default territory behaviors are noted. They occur when NLS_LANGUAGE is not specified.

<table>
<thead>
<tr>
<th>Oracle Territory</th>
<th>Oracle Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>BELGIUM</td>
<td>DUTCH (default)</td>
</tr>
<tr>
<td>BELGIUM</td>
<td>FRENCH</td>
</tr>
<tr>
<td>BELGIUM</td>
<td>GERMAN</td>
</tr>
<tr>
<td>CANADA</td>
<td>FRENCH (default)</td>
</tr>
<tr>
<td>CANADA</td>
<td>ENGLISH</td>
</tr>
<tr>
<td>DJIBOUTI</td>
<td>FRENCH (default)</td>
</tr>
<tr>
<td>DJIBOUTI</td>
<td>ARABIC</td>
</tr>
<tr>
<td>FINLAND</td>
<td>FINLAND (default)</td>
</tr>
<tr>
<td>FINLAND</td>
<td>SWEDISH</td>
</tr>
<tr>
<td>HONG KONG</td>
<td>TRADITIONAL CHINESE (default)</td>
</tr>
<tr>
<td>HONG KONG</td>
<td>ENGLISH</td>
</tr>
<tr>
<td>INDIA</td>
<td>ENGLISH (default)</td>
</tr>
</tbody>
</table>
Should the NLS_LANG Setting Match the Database Character Set?

The NLS_LANG character set should reflect the setting of the operating system character set of the client. For example, if the database character set is AL32UTF8 and the client is running on a Windows operating system, then you should not set AL32UTF8 as the client character set in the NLS_LANG parameter because there are no UTF-8 WIN32 clients. Instead the NLS_LANG setting should reflect the code page of the client. For example, on an English Windows client, the code page is 1252. An appropriate setting for NLS_LANG is AMERICAN_AMERICA.WE8MSWIN1252.

---

Table 3–3 Oracle Locale Variants (Cont.)

<table>
<thead>
<tr>
<th>Oracle Territory</th>
<th>Oracle Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDIA</td>
<td>ASSAMESE</td>
</tr>
<tr>
<td>INDIA</td>
<td>BANGLA</td>
</tr>
<tr>
<td>INDIA</td>
<td>GUJARATI</td>
</tr>
<tr>
<td>INDIA</td>
<td>HINDI</td>
</tr>
<tr>
<td>INDIA</td>
<td>KANNADA</td>
</tr>
<tr>
<td>INDIA</td>
<td>MALAYALAM</td>
</tr>
<tr>
<td>INDIA</td>
<td>MARATHI</td>
</tr>
<tr>
<td>INDIA</td>
<td>ORIYA</td>
</tr>
<tr>
<td>INDIA</td>
<td>PUNJABI</td>
</tr>
<tr>
<td>INDIA</td>
<td>TAMIL</td>
</tr>
<tr>
<td>INDIA</td>
<td>TELUGU</td>
</tr>
<tr>
<td>LUXEMBOURG</td>
<td>GERMAN (default)</td>
</tr>
<tr>
<td>LUXEMBOURG</td>
<td>FRENCH</td>
</tr>
<tr>
<td>SINGAPORE</td>
<td>ENGLISH (default)</td>
</tr>
<tr>
<td>SINGAPORE</td>
<td>MALAY</td>
</tr>
<tr>
<td>SINGAPORE</td>
<td>SIMPLIFIED CHINESE</td>
</tr>
<tr>
<td>SINGAPORE</td>
<td>TAMIL</td>
</tr>
<tr>
<td>SWITZERLAND</td>
<td>GERMAN (default)</td>
</tr>
<tr>
<td>SWITZERLAND</td>
<td>FRENCH</td>
</tr>
<tr>
<td>SWITZERLAND</td>
<td>ITALIAN</td>
</tr>
</tbody>
</table>
Setting NLS_LANG correctly allows proper conversion from the client operating system character set to the database character set. When these settings are the same, Oracle assumes that the data being sent or received is encoded in the same character set as the database character set, so no validation or conversion is performed. This can lead to corrupt data if the client code page and the database character set are different and conversions are necessary.

See Also: Oracle Database Installation Guide for Windows for more information about commonly used values of the NLS_LANG parameter in Windows

NLS Database Parameters

When a new database is created during the execution of the CREATE DATABASE statement, the NLS-related database configuration is established. The current NLS instance parameters are stored in the data dictionary along with the database and national character sets. The NLS instance parameters are read from the initialization parameter file at instance startup.

You can find the values for NLS parameters by using:

- NLS Data Dictionary Views
- NLS Dynamic Performance Views
- OCINlsGetInfo() Function

NLS Data Dictionary Views

Applications can check the session, instance, and database NLS parameters by querying the following data dictionary views:

- **NLS_SESSION_PARAMETERS** shows the NLS parameters and their values for the session that is querying the view. It does not show information about the character set.

- **NLS_INSTANCE_PARAMETERS** shows the current NLS instance parameters that have been explicitly set and the values of the NLS instance parameters.

- **NLS_DATABASE_PARAMETERS** shows the values of the NLS parameters for the database. The values are stored in the database.

NLS Dynamic Performance Views

Applications can check the following NLS dynamic performance views:
Language and Territory Parameters

- \texttt{V$NLS\_VALID\_VALUES} lists values for the following NLS parameters: \texttt{NLS\_LANGUAGE}, \texttt{NLS\_SORT}, \texttt{NLS\_TERRITORY}, \texttt{NLS\_CHARACTERSET}

- \texttt{V$NLS\_PARAMETERS} shows current values of the following NLS parameters: \texttt{NLS\_CALENDAR}, \texttt{NLS\_CHARACTERSET}, \texttt{NLS\_CURRENCY}, \texttt{NLS\_DATE\_FORMAT}, \texttt{NLS\_DATE\_LANGUAGE}, \texttt{NLS\_ISO\_CURRENCY}, \texttt{NLS\_LANGUAGE}, \texttt{NLS\_NUMERIC\_CHARACTERS}, \texttt{NLS\_SORT}, \texttt{NLS\_TERRITORY}, \texttt{NLS\_NCHAR\_CHARACTERSET}, \texttt{NLS\_COMP}, \texttt{NLS\_LENGTH\_SEMANTICS}, \texttt{NLS\_NCHAR\_CONV\_EXP}, \texttt{NLS\_TIMESTAMP\_FORMAT}, \texttt{NLS\_TIMESTAMP\_TZ\_FORMAT}, \texttt{NLS\_TIME\_FORMAT}, \texttt{NLS\_TIME\_TZ\_FORMAT}

See Also: "Oracle Database Reference"

\textbf{OCINlsGetInfo() Function}

User applications can query client NLS settings with the \texttt{OCINlsGetInfo()} function.

See Also: "Getting Locale Information in OCI" on page 10-3 for the description of \texttt{OCINlsGetInfo()}

\section*{Language and Territory Parameters}

This section contains information about the following parameters:

- \texttt{NLS\_LANGUAGE}
- \texttt{NLS\_TERRITORY}

\subsection*{NLS\_LANGUAGE}

\begin{tabular}{|l|l|}
\hline
\textbf{Property} & \textbf{Description} \\
\hline
Parameter type & String \\
Parameter scope & Initialization parameter and \texttt{ALTER SESSION} \\
Default value & Derived from \texttt{NLS\_LANG} \\
Range of values & Any valid language name \\
\hline
\end{tabular}

\texttt{NLS\_LANGUAGE} specifies the default conventions for the following session characteristics:
Language and Territory Parameters

- Language for server messages
- Language for day and month names and their abbreviations (specified in the SQL functions `TO_CHAR` and `TO_DATE`)
- Symbols for equivalents of AM, PM, AD, and BC. (A.M., P.M., A.D., and B.C. are valid only if `NLS_LANGUAGE` is set to `AMERICAN`.)
- Default sorting sequence for character data when `ORDER BY` is specified. (`GROUP BY` uses a binary sort unless `ORDER BY` is specified.)
- Writing direction
- Affirmative and negative response strings (for example, `YES` and `NO`)

The value specified for `NLS_LANGUAGE` in the initialization parameter file is the default for all sessions in that instance. For example, to specify the default session language as French, the parameter should be set as follows:

```
NLS_LANGUAGE = FRENCH
```

Consider the following server message:

```
ORA-00942: table or view does not exist
```

When the language is French, the server message appears as follows:

```
ORA-00942: table ou vue inexistante
```

Messages used by the server are stored in binary-format files that are placed in the `$ORACLE_HOME/product_name/mesg` directory, or the equivalent for your operating system. Multiple versions of these files can exist, one for each supported language, using the following filename convention:

```
<product_id><language_abbrev>.MSB
```

For example, the file containing the server messages in French is called `oraf.msb`, because `ORA` is the product ID (`<product_id>`) and `F` is the language abbreviation (`<language_abbrev>`) for French. The `product_name` is `rdbms`, so it is in the `$ORACLE_HOME/rdbms/mesg` directory.

If `NLS_LANG` is specified in the client environment, then the value of `NLS_LANGUAGE` in the initialization parameter file is overridden at connection time.

Messages are stored in these files in one specific character set, depending on the language and the operating system. If this character set is different from the database character set, then message text is automatically converted to the database character set. If necessary, it is then converted to the client character set if the client
character set is different from the database character set. Hence, messages are
displayed correctly at the user's terminal, subject to the limitations of character set
conversion.

The language-specific binary message files that are actually installed depend on the
languages that the user specifies during product installation. Only the English
binary message file and the language-specific binary message files specified by the
user are installed.

The default value of NLS_LANGUAGE may be specific to the operating system. You
can alter the NLS_LANGUAGE parameter by changing its value in the initialization
parameter file and then restarting the instance.

See Also: Your operating system-specific Oracle documentation
for more information about the default value of NLS_LANGUAGE

All messages and text should be in the same language. For example, when you run
an Oracle Developer application, the messages and boilerplate text that you see
originate from three sources:

- Messages from the server
- Messages and boilerplate text generated by Oracle Forms
- Messages and boilerplate text generated by the application

NLS_LANGUAGE determines the language used for the first two kinds of text. The
application is responsible for the language used in its messages and boilerplate text.
The following examples show behavior that results from setting NLS_LANGUAGE to
different values.

Example 3–3 NLS_LANGUAGE=ITALIAN

Use the ALTER SESSION statement to set NLS_LANGUAGE to Italian:

ALTER SESSION SET NLS_LANGUAGE=Italian;

Enter a SELECT statement:

SQL> SELECT last_name, hire_date, ROUND(salary/8,2) salary FROM employees;

You should see results similar to the following:

<table>
<thead>
<tr>
<th>LAST_NAME</th>
<th>HIRE_DATE</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sciarra</td>
<td>30-SET-97</td>
<td>962.5</td>
</tr>
</tbody>
</table>
Note that the month name abbreviations are in Italian.

**See Also:** "Overriding Default Values for NLS_LANGUAGE and NLS_TERRITORY During a Session" on page 3-18 for more information about using the ALTER SESSION statement.

**Example 3-4  NLS_LANGUAGE=GERMAN**

Use the ALTER SESSION statement to change the language to German:

```sql
SQL> ALTER SESSION SET NLS_LANGUAGE=German;
```

Enter the same SELECT statement:

```sql
SQL> SELECT last_name, hire_date, ROUND(salary/8,2) salary FROM employees;
```

You should see results similar to the following:

<table>
<thead>
<tr>
<th>LAST_NAME</th>
<th>HIRE_DATE</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sciarra</td>
<td>30-SEP-97</td>
<td>962.5</td>
</tr>
<tr>
<td>Urman</td>
<td>07-MÄR-98</td>
<td>975</td>
</tr>
<tr>
<td>Popp</td>
<td>07-DEZ-99</td>
<td>862.5</td>
</tr>
</tbody>
</table>

Note that the language of the month abbreviations has changed.

**NLS_TERRITORY**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter scope</td>
<td>Initialization parameter and ALTER SESSION</td>
</tr>
<tr>
<td>Default value</td>
<td>Derived from NLS_LANG</td>
</tr>
<tr>
<td>Range of values</td>
<td>Any valid territory name</td>
</tr>
</tbody>
</table>

NLS_TERRITORY specifies the conventions for the following default date and numeric formatting characteristics:

- Date format
Language and Territory Parameters

- Decimal character and group separator
- Local currency symbol
- ISO currency symbol
- Dual currency symbol
- First day of the week
- Credit and debit symbols
- ISO week flag
- List separator

The value specified for NLS_TERRITORY in the initialization parameter file is the default for the instance. For example, to specify the default as France, the parameter should be set as follows:

```
NLS_TERRITORY = FRANCE
```

When the territory is FRANCE, numbers are formatted using a comma as the decimal character.

You can alter the NLS_TERRITORY parameter by changing the value in the initialization parameter file and then restarting the instance. The default value of NLS_TERRITORY can be specific to the operating system.

If NLS_LANG is specified in the client environment, then the value of NLS_TERRITORY in the initialization parameter file is overridden at connection time.

The territory can be modified dynamically during the session by specifying the new NLS_TERRITORY value in an ALTER SESSION statement. Modifying NLS_TERRITORY resets all derived NLS session parameters to default values for the new territory.

To change the territory to France during a session, issue the following ALTER SESSION statement:

```
ALTER SESSION SET NLS_TERRITORY = France;
```

The following examples show behavior that results from different settings of NLS_TERRITORY and NLS_LANGUAGE.

**Example 3-5  NLS_LANGUAGE=AMERICAN, NLS_TERRITORY=AMERICA**

Enter the following SELECT statement:

```
SQL> SELECT TO_CHAR(salary,'$99G999D99') salary FROM employees;
```
When NLS_TERRITORY is set to AMERICA and NLS_LANGUAGE is set to AMERICAN, results similar to the following should appear:

```
SALARY
--------------
$24,000.00
$17,000.00
$17,000.00
```

**Example 3–6  NLS_LANGUAGE=AMERICAN, NLS_TERRITORY=GERMANY**

Use an ALTER SESSION statement to change the territory to Germany:

```
ALTER SESSION SET NLS_TERRITORY = Germany;
Session altered.
```

Enter the same SELECT statement as before:

```
SQL> SELECT TO_CHAR(salary,'L99G999D99') salary FROM employees;
```

You should see results similar to the following:

```
SALARY
--------------
€24,000.00
€17,000.00
€17,000.00
```

Note that the currency symbol has changed from $ to €. The numbers have not changed because the underlying data is the same.

**See Also:** "Overriding Default Values for NLS_LANGUAGE and NLS_TERRITORY During a Session" on page 3-18 for more information about using the ALTER SESSION statement

**Example 3–7  NLS_LANGUAGE=GERMAN, NLS_TERRITORY=GERMANY**

Use an ALTER SESSION statement to change the language to German:

```
ALTER SESSION SET NLS_LANGUAGE = German;
Sitzung wurde geändert.
```

Note that the server message now appears in German.

Enter the same SELECT statement as before:
SQL> SELECT TO_CHAR(salary, 'L99G999D99') salary FROM employees;

You should see the same results as in Example 3–6:

<table>
<thead>
<tr>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>€24.000,00</td>
</tr>
<tr>
<td>€17.000,00</td>
</tr>
<tr>
<td>€17.000,00</td>
</tr>
</tbody>
</table>

**Example 3–8 NLS_LANGUAGE=GERMAN, NLS_TERRITORY=AMERICA**

Use an ALTER SESSION statement to change the territory to America:

```sql
ALTER SESSION SET NLS_TERRITORY = America;
```

Sitzung wurde geändert.

Enter the same SELECT statement as in the other examples:

```sql
SQL> SELECT TO_CHAR(salary, 'L99G999D99') salary FROM employees;
```

You should see output similar to the following:

<table>
<thead>
<tr>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>$24,000.00</td>
</tr>
<tr>
<td>$17,000.00</td>
</tr>
<tr>
<td>$17,000.00</td>
</tr>
</tbody>
</table>

Note that the currency symbol changed from € to $ because the territory changed from Germany to America.

**Overriding Default Values for NLS_LANGUAGE and NLS_TERRITORY During a Session**

Default values for NLS_LANGUAGE and NLS_TERRITORY and default values for specific formatting parameters can be overridden during a session by using the ALTER SESSION statement.

**Example 3–9 NLS_LANG=ITALIAN_ITALY.WE8DEC**

Set the NLS_LANG environment variable so that the language is Italian, the territory is Italy, and the character set is WE8DEC:

```bash
% setenv NLS_LANG Italian_Italy.WE8DEC
```

Enter a SELECT statement:
You should see output similar to the following:

<table>
<thead>
<tr>
<th>LAST_NAME</th>
<th>HIRE_DATE</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sciarr</td>
<td>30-SEP-97</td>
<td>962,5</td>
</tr>
<tr>
<td>Urman</td>
<td>07-MAR-98</td>
<td>975</td>
</tr>
<tr>
<td>Popp</td>
<td>07-DEC-99</td>
<td>862,5</td>
</tr>
</tbody>
</table>

Note the language of the month abbreviations and the decimal character.

**Example 3–10  Change Language, Date Format, and Decimal Character**

Use ALTER SESSION statements to change the language, the date format, and the decimal character:

```sql
SQL> ALTER SESSION SET NLS_LANGUAGE=german;
Session wurde geändert.

SQL> ALTER SESSION SET NLS_DATE_FORMAT='DD.MON.YY';
Session wurde geändert.

SQL> ALTER SESSION SET NLS_NUMERIC_CHARACTERS='.,';
Session wurde geändert.
```

Enter the SELECT statement shown in Example 3–9:

```sql
SQL> SELECT last_name, hire_date, ROUND(salary/8,2) salary FROM employees;
```

You should see output similar to the following:

<table>
<thead>
<tr>
<th>LAST_NAME</th>
<th>HIRE_DATE</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sciarr</td>
<td>30.SEP.97</td>
<td>962.5</td>
</tr>
<tr>
<td>Urman</td>
<td>07.MAR.98</td>
<td>975</td>
</tr>
<tr>
<td>Popp</td>
<td>07.DEZ.99</td>
<td>862.5</td>
</tr>
</tbody>
</table>

Note that the language of the month abbreviations is German and the decimal character is a period.

The behavior of the NLS_LANG environment variable implicitly determines the language environment of the database for each session. When a session connects to
a database, an ALTER SESSION statement is automatically executed to set the values of the database parameters NLS_LANGUAGE and NLS_TERRITORY to those specified by the language and territory arguments of NLS_LANG. If NLS_LANG is not defined, then no implicit ALTER SESSION statement is executed.

When NLS_LANG is defined, the implicit ALTER SESSION is executed for all instances to which the session connects, for both direct and indirect connections. If the values of NLS parameters are changed explicitly with ALTER SESSION during a session, then the changes are propagated to all instances to which that user session is connected.

Date and Time Parameters

Oracle enables you to control the display of date and time. This section contains the following topics:

- Date Formats
- Time Formats

Date Formats

Different date formats are shown in Table 3–4.

<table>
<thead>
<tr>
<th>Country</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estonia</td>
<td>dd.mm.yyyy</td>
<td>28.02.2003</td>
</tr>
<tr>
<td>Germany</td>
<td>dd-mm-rr</td>
<td>28-02-03</td>
</tr>
<tr>
<td>Japan</td>
<td>rr-mm-dd</td>
<td>03-02-28</td>
</tr>
<tr>
<td>UK</td>
<td>dd-mon-rr</td>
<td>28-Feb-03</td>
</tr>
<tr>
<td>US</td>
<td>dd-mon-rr</td>
<td>28-Feb-03</td>
</tr>
</tbody>
</table>

This section includes the following parameters:

- NLS_DATE_FORMAT
- NLS_DATE_LANGUAGE
**NLS_DATE_FORMAT**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter scope</td>
<td>Initialization parameter, environment variable, and ALTER SESSION</td>
</tr>
<tr>
<td>Default value</td>
<td>Derived from NLS_TERRITORY</td>
</tr>
<tr>
<td>Range of values</td>
<td>Any valid date format mask</td>
</tr>
</tbody>
</table>

The NLS_DATE_FORMAT parameter defines the default date format to use with the TO_CHAR and TO_DATE functions. The NLS_TERRITORY parameter determines the default value of NLS_DATE_FORMAT. The value of NLS_DATE_FORMAT can be any valid date format mask. For example:

\[
\text{NLS\_DATE\_FORMAT} = \"MM/DD/YYYY\"
\]

To add string literals to the date format, enclose the string literal with double quotes. Note that when double quotes are included in the date format, the entire value must be enclosed by single quotes. For example:

\[
\text{NLS\_DATE\_FORMAT} = \\"\text{Date: } \text{MM/DD/YYYY}\\"
\]

**Example 3–11 Setting the Date Format to Display Roman Numerals**

To set the default date format to display Roman numerals for the month, include the following line in the initialization parameter file:

\[
\text{NLS\_DATE\_FORMAT} = \"DD RM YYYY\"
\]

Enter the following SELECT statement:

\[
\text{SELECT TO\_CHAR(SYSDATE) currdate FROM dual;}
\]

You should see the following output if today’s date is February 12, 1997:

\[
\text{CURRDATE} \\
\text{--------} \\
12 II 1997
\]

The value of NLS_DATE_FORMAT is stored in the internal date format. Each format element occupies two bytes, and each string occupies the number of bytes in the string plus a terminator byte. Also, the entire format mask has a two-byte
Date and Time Parameters

terminator. For example, "MM/DD/YY" occupies 14 bytes internally because there are three format elements (month, day, and year), two 3-byte strings (the two slashes), and the two-byte terminator for the format mask. The format for the value of NLS_DATE_FORMAT cannot exceed 24 bytes.

You can alter the default value of NLS_DATE_FORMAT by:

- Changing its value in the initialization parameter file and then restarting the instance
- Using an ALTER SESSION SET NLS_DATE_FORMAT statement

See Also: Oracle Database SQL Reference for more information about date format elements and the ALTER SESSION statement

If a table or index is partitioned on a date column, and if the date format specified by NLS_DATE_FORMAT does not specify the first two digits of the year, then you must use the TO_DATE function with a 4-character format mask for the year.

For example:

TO_DATE('11-jan-1997', 'dd-mon-yyyy')

See Also: Oracle Database SQL Reference for more information about partitioning tables and indexes and using TO_DATE

### NLS_DATE_LANGUAGE

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter scope</td>
<td>Initialization parameter, environment variable, ALTER SESSION, and SQL functions</td>
</tr>
<tr>
<td>Default value</td>
<td>Derived from NLS_LANGUAGE</td>
</tr>
<tr>
<td>Range of values</td>
<td>Any valid language name</td>
</tr>
</tbody>
</table>

The NLS_DATE_LANGUAGE parameter specifies the language for the day and month names produced by the TO_CHAR and TO_DATE functions. NLS_DATE_LANGUAGE overrides the language that is specified implicitly by NLS_LANGUAGE. NLS_DATE_LANGUAGE has the same syntax as the NLS_LANGUAGE parameter, and all supported languages are valid values.
NLS_DATE_LANGUAGE also determines the language used for:

- Month and day abbreviations returned by the TO_CHAR and TO_DATE functions
- Month and day abbreviations used by the default date format (NLS_DATE_FORMAT)
- Abbreviations for AM, PM, AD, and BC

**Example 3–12  NLS_DATE_LANGUAGE=FRENCH, Month and Day Names**

Set the date language to French:

```
ALTER SESSION SET NLS_DATE_LANGUAGE = FRENCH
```

Enter a SELECT statement:

```
SELECT TO_CHAR(SYSDATE, 'Day:DD Month yyyy') FROM dual;
```

You should see output similar to the following:

```
TO_CHAR(SYSDATE,'DAY:DDMONTHYYYY')
------------------------------------------------------------
Vendredi:07 Décembre  2001
```

When numbers are spelled in words using the TO_CHAR function, the English spelling is always used. For example, enter the following SELECT statement:

```
SQL> SELECT TO_CHAR(TO_DATE('12-Oct-2001'),'Day: ddspth Month') FROM dual;
```

You should see output similar to the following:

```
TO_CHAR(TO_DATE('12-OCT-2001'),'DAY:DDSPTHMONTH')
--------------------------------------------------------------------
Vendredi: twelfth Octobre
```

**Example 3–13  NLS_DATE_LANGUAGE=FRENCH, Month and Day Abbreviations**

Month and day abbreviations are determined by NLS_DATE_LANGUAGE. Enter the following SELECT statement:

```
SELECT TO_CHAR(SYSDATE, 'Dy:dd Mon yyyy') FROM dual;
```

You should see output similar to the following:

```
TO_CHAR(SYSDATE, 'DY:DDMO
------------------------
```
Example 3–14  NLS_DATE_LANGUAGE=FRENCH, Default Date Format
The default date format uses the month abbreviations determined by NLS_DATE_ LANGUAGE. For example, if the default date format is DD–MON–YYYY, then insert a date as follows:

```
INSERT INTO tablename VALUES ('12-Fév-1997');
```

See Also: Oracle Database SQL Reference

Time Formats
Different time formats are shown in Table 3–5.

Table 3–5  Time Formats

<table>
<thead>
<tr>
<th>Country</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Germany</td>
<td>hh24:mi:ss</td>
<td>13:50:23</td>
</tr>
<tr>
<td>Japan</td>
<td>hh24:mi:ss</td>
<td>13:50:23</td>
</tr>
<tr>
<td>UK</td>
<td>hh24:mi:ss</td>
<td>13:50:23</td>
</tr>
<tr>
<td>US</td>
<td>hh:mi:ssxff am</td>
<td>1:50:23.555 PM</td>
</tr>
</tbody>
</table>

See Also: Chapter 4, "Datetime Datatypes and Time Zone Support"

NLS_TIMESTAMP_FORMAT

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter scope</td>
<td>Initialization parameter, environment variable, and ALTER SESSION</td>
</tr>
<tr>
<td>Default value</td>
<td>Derived from NLS_TERRITORY</td>
</tr>
</tbody>
</table>
NLS_TIMESTAMP_FORMAT defines the default date format for the TIMESTAMP and TIMESTAMP WITH LOCAL TIME ZONE datatypes. The following example shows a value for NLS_TIMESTAMP_FORMAT:

```
NLS_TIMESTAMP_FORMAT = 'YYYY-MM-DD HH:MI:SS.FF'
```

**Example 3–15  Timestamp Format**

```
SQL> SELECT TO_TIMESTAMP('11-nov-2000 01:00:00.336', 'dd-mon-yyyy hh:mi:ss.ff')
    FROM dual;
```

You should see output similar to the following:

```
TO_TIMESTAMP('11-NOV-200001:00:00.336','DD-MON-YYYYHH:MI:SS.FF')
-----------------------------------------------
11-NOV-00 01:00:00.336000000
```

You can specify the value of NLS_TIMESTAMP_FORMAT by setting it in the initialization parameter file. You can specify its value for a client as a client environment variable.

You can also alter the value of NLS_TIMESTAMP_FORMAT by:

- Changing its value in the initialization parameter file and then restarting the instance
- Using the `ALTER SESSION SET NLS_TIMESTAMP_FORMAT` statement

**See Also:** Oracle Database SQL Reference for more information about the TO_TIMESTAMP function and the ALTER SESSION statement

### NLS_TIMESTAMP_TZ_FORMAT

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter scope</td>
<td>Initialization parameter, environment variable, and ALTER SESSION</td>
</tr>
<tr>
<td>Default value</td>
<td>Derived from NLS_TERRITORY</td>
</tr>
</tbody>
</table>
NLS_TIMESTAMP_TZ_FORMAT defines the default date format for the TIMESTAMP and TIMESTAMPT WITH LOCAL TIME ZONE datatypes. It is used with the TO_CHAR and TO_TIMESTAMP_TZ functions.

You can specify the value of NLS_TIMESTAMP_TZ_FORMAT by setting it in the initialization parameter file. You can specify its value for a client as a client environment variable.

**Example 3–16 Setting NLS_TIMESTAMP_TZ_FORMAT**

The format value must be surrounded by quotation marks. For example:

```
NLS_TIMESTAMP_TZ_FORMAT = 'YYYY-MM-DD HH:MI:SS.FF TZH:TZM'
```

The following example of the TO_TIMESTAMP_TZ function uses the format value that was specified for NLS_TIMESTAMP_TZ_FORMAT:

```
SQL> SELECT TO_TIMESTAMP_TZ('2000-08-20,05:00:00.55America/Los_Angeles', 'yyyy-mm-dd hh:mi:ss.ff TZR') FROM dual;
```

You should see output similar to the following:

```
TO_TIMESTAMP_TZ('2000-08-20,05:00:00.55America/Los_Angeles','YYYY-MM-DDHH:MI:SS.FF TZR')
```

You can change the value of NLS_TIMESTAMP_TZ_FORMAT by:

- Changing its value in the initialization parameter file and then restarting the instance
- Using the ALTER SESSION statement.
Calendar Definitions

See Also:

- Oracle Database SQL Reference for more information about the TO_TIMESTAMP_TZ function and the ALTER SESSION statement
- "Choosing a Time Zone File" on page 4-20 for more information about time zones

This section includes the following topics:

- Calendar Formats
- NLSCALENDAR

Calendar Formats

The following calendar information is stored for each territory:

- First Day of the Week
- First Calendar Week of the Year
- Number of Days and Months in a Year
- First Year of Era

First Day of the Week

Some cultures consider Sunday to be the first day of the week. Others consider Monday to be the first day of the week. A German calendar starts with Monday, as shown in Table 3–6.

<table>
<thead>
<tr>
<th>Mo</th>
<th>Di</th>
<th>Mi</th>
<th>Do</th>
<th>Fr</th>
<th>Sa</th>
<th>So</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
</tr>
</tbody>
</table>
The first day of the week is determined by the NLS_TERRITORY parameter.

**See Also:** "NLS_TERRITORY" on page 3-15

**First Calendar Week of the Year**

Some countries use week numbers for scheduling, planning, and bookkeeping. Oracle supports this convention. In the ISO standard, the week number can be different from the week number of the calendar year. For example, 1st Jan 1988 is in ISO week number 53 of 1987. An ISO week always starts on a Monday and ends on a Sunday.

- If January 1 falls on a Friday, Saturday, or Sunday, then the ISO week that includes January 1 is the last week of the previous year, because most of the days in the week belong to the previous year.
- If January 1 falls on a Monday, Tuesday, Wednesday, or Thursday, then the ISO week is the first week of the new year, because most of the days in the week belong to the new year.

To support the ISO standard, Oracle provides the IW date format element. It returns the ISO week number.

**Table 3-7** shows an example in which January 1 occurs in a week that has four or more days in the first calendar week of the year. The week containing January 1 is the first ISO week of 1998.

**Table 3-7 First ISO Week of the Year: Example 1, January 1998**

<table>
<thead>
<tr>
<th>Mo</th>
<th>Tu</th>
<th>We</th>
<th>Th</th>
<th>Fr</th>
<th>Sa</th>
<th>Su</th>
<th>ISO Week</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>First ISO week of 1998</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>Second ISO week of 1998</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>Third ISO week of 1998</td>
</tr>
<tr>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>-</td>
<td>Fifth ISO week of 1998</td>
</tr>
</tbody>
</table>

---

3-28 Oracle Database Globalization Support Guide
Table 3–8 shows an example in which January 1 occurs in a week that has three or fewer days in the first calendar week of the year. The week containing January 1 is the 53rd ISO week of 1998, and the following week is the first ISO week of 1999.

<table>
<thead>
<tr>
<th>Mo</th>
<th>Tu</th>
<th>We</th>
<th>Th</th>
<th>Fr</th>
<th>Sa</th>
<th>Su</th>
<th>ISO Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>Fifty-third ISO week of 1998</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>First ISO week of 1999</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>Second ISO week of 1999</td>
</tr>
<tr>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>Third ISO week of 1999</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>Fourth ISO week of 1999</td>
</tr>
</tbody>
</table>

The first calendar week of the year is determined by the NLS_TERRITORY parameter.

See Also: "NLS_TERRITORY" on page 3-15

Number of Days and Months in a Year

Oracle supports six calendar systems in addition to Gregorian, the default:

- Japanese Imperial—uses the same number of months and days as Gregorian, but the year starts with the beginning of each Imperial Era
- ROC Official—uses the same number of months and days as Gregorian, but the year starts with the founding of the Republic of China
- Persian—has 31 days for each of the first six months. The next five months have 30 days each. The last month has either 29 days or 30 days (leap year).
- Thai Buddha—uses a Buddhist calendar
- Arabic Hijrah—has 12 months with 354 or 355 days
- English Hijrah—has 12 months with 354 or 355 days

The calendar system is specified by the NLS_CALENDAR parameter.

See Also: "NLS_CALENDAR" on page 3-30

First Year of Era

The Islamic calendar starts from the year of the Hegira.
The Japanese Imperial calendar starts from the beginning of an Emperor's reign. For example, 1998 is the tenth year of the Heisei era. It should be noted, however, that the Gregorian system is also widely understood in Japan, so both 98 and Heisei 10 can be used to represent 1998.

**NLS_CALENDAR**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter scope</td>
<td>Initialization parameter, environment variable, ALTER SESSION, and SQL functions</td>
</tr>
<tr>
<td>Default value</td>
<td>Gregorian</td>
</tr>
<tr>
<td>Range of values</td>
<td>Any valid calendar format name</td>
</tr>
</tbody>
</table>

Many different calendar systems are in use throughout the world. NLS_CALENDAR specifies which calendar system Oracle uses.

NLS_CALENDAR can have one of the following values:

- Arabic Hijrah
- English Hijrah
- Gregorian
- Japanese Imperial
- Persian
- ROC Official (Republic of China)
- Thai Buddha

**See Also:** Appendix A, "Locale Data" for a list of calendar systems, their default date formats, and the character sets in which dates are displayed

**Example 3–17**  

NLS_CALENDAR='English Hijrah'

Set NLS_CALENDAR to English Hijrah.

```
SQL> ALTER SESSION SET NLS_CALENDAR='English Hijrah';
```

Enter a SELECT statement to display SYSDATE:
SELECT SYSDATE FROM dual;

You should see output similar to the following:

SYSDATE
--------------------
24 Ramadan     1422

Numeric and List Parameters

This section includes the following topics:

- Numeric Formats
- NLS_NUMERIC_CHARACTERS
- NLS_LIST_SEPARATOR

Numeric Formats

The database must know the number-formatting convention used in each session to interpret numeric strings correctly. For example, the database needs to know whether numbers are entered with a period or a comma as the decimal character (234.00 or 234,00). Similarly, applications must be able to display numeric information in the format expected at the client site.

Examples of numeric formats are shown in Table 3–9.

<table>
<thead>
<tr>
<th>Country</th>
<th>Numeric Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estonia</td>
<td>1 234 567,89</td>
</tr>
<tr>
<td>Germany</td>
<td>1.234,567,89</td>
</tr>
<tr>
<td>Japan</td>
<td>1,234,567.89</td>
</tr>
<tr>
<td>UK</td>
<td>1,234,567.89</td>
</tr>
<tr>
<td>US</td>
<td>1,234,567.89</td>
</tr>
</tbody>
</table>

Numeric formats are derived from the setting of the NLS_TERRITORY parameter, but they can be overridden by the NLS_NUMERIC_CHARACTERS parameter.

See Also: "NLS_TERRITORY" on page 3-15
NLS_NUMERIC_CHARACTERS

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter scope</td>
<td>Initialization parameter, environment variable, ALTER SESSION, and SQL functions</td>
</tr>
<tr>
<td>Default value</td>
<td>Default decimal character and group separator for a particular territory</td>
</tr>
<tr>
<td>Range of values</td>
<td>Any two valid numeric characters</td>
</tr>
</tbody>
</table>

This parameter specifies the decimal character and group separator. The group separator is the character that separates integer groups to show thousands and millions, for example. The group separator is the character returned by the G number format mask. The decimal character separates the integer and decimal parts of a number. Setting NLS_NUMERIC_CHARACTERS overrides the values derived from the setting of NLS_TERRITORY.

Any character can be the decimal character or group separator. The two characters specified must be single-byte, and the characters must be different from each other. The characters cannot be any numeric character or any of the following characters: plus (+), hyphen (-), less than sign (<), greater than sign (>). Either character can be a space.

Example 3–18 Setting NLS_NUMERIC_CHARACTERS

To set the decimal character to a comma and the grouping separator to a period, define NLS_NUMERIC_CHARACTERS as follows:

```sql
ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ",."
```

SQL statements can include numbers represented as numeric or text literals. Numeric literals are not enclosed in quotes. They are part of the SQL language syntax and always use a dot as the decimal character and never contain a group separator. Text literals are enclosed in single quotes. They are implicitly or explicitly converted to numbers, if required, according to the current NLS settings.

The following SELECT statement formats the number 4000 with the decimal character and group separator specified in the ALTER SESSION statement:

```sql
SELECT TO_CHAR(4000, '9G999D99') FROM dual;
```

You should see output similar to the following:
You can change the default value of `NLS_NUMERIC_CHARACTERS` by:

- Changing the value of `NLS_NUMERIC_CHARACTERS` in the initialization parameter file and then restart the instance
- Using the `ALTER SESSION` statement to change the parameter's value during a session

See Also: *Oracle Database SQL Reference* for more information about the `ALTER SESSION` statement

### NLS_LIST_SEPARATOR

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter scope</td>
<td>Environment variable</td>
</tr>
<tr>
<td>Default value</td>
<td>Derived from <code>NLS_TERRITORY</code></td>
</tr>
<tr>
<td>Range of values</td>
<td>Any valid character</td>
</tr>
</tbody>
</table>

`NLS_LIST_SEPARATOR` specifies the character to use to separate values in a list of values (usually `,` or `. `or` ;` or `:`). Its default value is derived from the value of `NLS_TERRITORY`. For example, a list of numbers from 1 to 5 can be expressed as `1,2,3,4,5` or `1.2.3.4.5` or `1;2;3;4;5` or `1:2:3:4:5`.

The character specified must be single-byte and cannot be the same as either the numeric or monetary decimal character, any numeric character, or any of the following characters: plus (+), hyphen (-), less than sign (<), greater than sign (>), period (.).

### Monetary Parameters

This section includes the following topics:

- Currency Formats
- NLS_CURRENCY
Monetary Parameters

- **NLS_ISO_CURRENCY**
- **NLS_DUAL_CURRENCY**
- **NLS_MONETARY_CHARACTERS**
- **NLS_CREDIT**
- **NLS_DEBIT**

**Currency Formats**

Different currency formats are used throughout the world. Some typical ones are shown in Table 3–10.

<table>
<thead>
<tr>
<th>Country</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estonia</td>
<td>1 234,56 kr</td>
</tr>
<tr>
<td>Germany</td>
<td>1.234,56€</td>
</tr>
<tr>
<td>Japan</td>
<td>¥1,234.56</td>
</tr>
<tr>
<td>UK</td>
<td>£1,234.56</td>
</tr>
<tr>
<td>US</td>
<td>$1,234.56</td>
</tr>
</tbody>
</table>

**Table 3–10  Currency Format Examples**

**NLS_CURRENCY**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter scope</td>
<td>Initialization parameter, environment variable, ALTER SESSION, and SQL functions</td>
</tr>
<tr>
<td>Default value</td>
<td>Derived from NLS_TERRITORY</td>
</tr>
<tr>
<td>Range of values</td>
<td>Any valid currency symbol string</td>
</tr>
</tbody>
</table>

NLS_CURRENCY specifies the character string returned by the L number format mask, the local currency symbol. Setting NLS_CURRENCY overrides the setting defined implicitly by NLS_TERRITORY.
Example 3–19  Displaying the Local Currency Symbol

Connect to the sample order entry schema:

```
SQL> connect oe/oe
Connected.
```

Enter a SELECT statement similar to the following:

```
SQL> SELECT TO_CHAR(order_total, 'L099G999D99') "total" FROM orders
    WHERE order_id > 2450;
```

You should see output similar to the following:

```
  total
  ---------------------
     $078,279.60
     $006,653.40
     $014,087.50
     $010,474.60
     $012,589.00
     $000,129.00
     $003,878.40
     $021,586.20
```

You can change the default value of `NLS_CURRENCY` by:

- Changing its value in the initialization parameter file and then restarting the instance
- Using an ALTER SESSION statement

See Also: Oracle Database SQL Reference for more information about the ALTER SESSION statement

NLS_ISO_CURRENCY

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter scope</td>
<td>Initialization parameter, environment variable, ALTER SESSION, and SQL functions</td>
</tr>
<tr>
<td>Default value</td>
<td>Derived from NLS_TERRITORY</td>
</tr>
<tr>
<td>Range of values</td>
<td>Any valid string</td>
</tr>
</tbody>
</table>
NLS_ISO_CURRENCY specifies the character string returned by the C number format mask, the ISO currency symbol. Setting NLS_ISO_CURRENCY overrides the value defined implicitly by NLS_TERRITORY.

Local currency symbols can be ambiguous. For example, a dollar sign ($) can refer to US dollars or Australian dollars. ISO specifications define unique currency symbols for specific territories or countries. For example, the ISO currency symbol for the US dollar is USD. The ISO currency symbol for the Australian dollar is AUD.

More ISO currency symbols are shown in Table 3–11.

<table>
<thead>
<tr>
<th>Country</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estonia</td>
<td>1 234 567,89 EEK</td>
</tr>
<tr>
<td>Germany</td>
<td>1.234,567,89 EUR</td>
</tr>
<tr>
<td>Japan</td>
<td>1,234,567.89 JPY</td>
</tr>
<tr>
<td>UK</td>
<td>1,234,567.89 GBP</td>
</tr>
<tr>
<td>US</td>
<td>1,234,567.89 USD</td>
</tr>
</tbody>
</table>

NLS_ISO_CURRENCY has the same syntax as the NLS_TERRITORY parameter, and all supported territories are valid values.

**Example 3–20  Setting NLS_ISO_CURRENCY**

This example assumes that you are connected as oe/oe in the sample schema.

To specify the ISO currency symbol for France, set NLS_ISO_CURRENCY as follows:

```
ALTER SESSION SET NLS_ISO_CURRENCY = FRANCE;
```

Enter a SELECT statement:

```
SQL> SELECT TO_CHAR(order_total, 'C099G999D99') "TOTAL" FROM orders
WHERE customer_id = 146;
```

You should see output similar to the following:

```
TOTAL
------------------
EUR017,848.20
EUR027,455.30
EUR029,249.10
EUR013,824.00
```
You can change the default value of \texttt{NLS\_ISO\_CURRENCY} by:

- Changing its value in the initialization parameter file and then restarting the instance
- Using an \texttt{ALTER SESSION} statement

\textbf{See Also:} \textit{Oracle Database SQL Reference} for more information about the \texttt{ALTER SESSION} statement

\textbf{NLS\_DUAL\_CURRENCY}

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter scope</td>
<td>Initialization parameter, environmental variable, \texttt{ALTER SESSION}, and SQL functions</td>
</tr>
<tr>
<td>Default value</td>
<td>Derived from \texttt{NLS_TERRITORY}</td>
</tr>
<tr>
<td>Range of values</td>
<td>Any valid symbol</td>
</tr>
</tbody>
</table>

Use \texttt{NLS\_DUAL\_CURRENCY} to override the default dual currency symbol defined implicitly by \texttt{NLS\_TERRITORY}.

\texttt{NLS\_DUAL\_CURRENCY} was introduced to support the euro currency symbol during the euro transition period. Table 3–12 lists the character sets that support the euro symbol.

\textbf{Table 3–12 Character Sets that Support the Euro Symbol}

<table>
<thead>
<tr>
<th>Character Set Name</th>
<th>Description</th>
<th>Hexadecimal Code Value of the Euro Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>D8EBCDIC1141</td>
<td>EBCDIC Code Page 1141 8-bit Austrian German</td>
<td>9F</td>
</tr>
<tr>
<td>DK8EBCDIC1142</td>
<td>EBCDIC Code Page 1142 8-bit Danish</td>
<td>5A</td>
</tr>
<tr>
<td>S8EBCDIC1143</td>
<td>EBCDIC Code Page 1143 8-bit Swedish</td>
<td>5A</td>
</tr>
<tr>
<td>I8EBCDIC1144</td>
<td>EBCDIC Code Page 1144 8-bit Italian</td>
<td>9F</td>
</tr>
<tr>
<td>F8EBCDIC1147</td>
<td>EBCDIC Code Page 1147 8-bit French</td>
<td>9F</td>
</tr>
<tr>
<td>WE8PC858</td>
<td>IBM-PC Code Page 858 8-bit West European</td>
<td>DF</td>
</tr>
<tr>
<td>WE8ISO8859P15</td>
<td>ISO 8859-15 West European</td>
<td>A4</td>
</tr>
</tbody>
</table>
## Table 3–12 Character Sets that Support the Euro Symbol (Cont.)

<table>
<thead>
<tr>
<th>Character Set Name</th>
<th>Description</th>
<th>Hexadecimal Code Value of the Euro Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>EE8MSWIN1250</td>
<td>MS Windows Code Page 1250 8-bit East European</td>
<td>80</td>
</tr>
<tr>
<td>CL8MSWIN1251</td>
<td>MS Windows Code Page 1251 8-bit Latin/Cyrillic</td>
<td>88</td>
</tr>
<tr>
<td>WE8MSWIN1252</td>
<td>MS Windows Code Page 1252 8-bit West European</td>
<td>80</td>
</tr>
<tr>
<td>EL8MSWIN1253</td>
<td>MS Windows Code Page 1253 8-bit Latin/Greek</td>
<td>80</td>
</tr>
<tr>
<td>WE8EBCDIC1047E</td>
<td>Latin 1/Open Systems 1047</td>
<td>9F</td>
</tr>
<tr>
<td>WE8EBCDIC1140</td>
<td>EBCDIC Code Page 1140 8-bit West European</td>
<td>9F</td>
</tr>
<tr>
<td>WE8EBCDIC1140C</td>
<td>EBCDIC Code Page 1140 Client 8-bit West European</td>
<td>9F</td>
</tr>
<tr>
<td>WE8EBCDIC1145</td>
<td>EBCDIC Code Page 1145 8-bit West European</td>
<td>9F</td>
</tr>
<tr>
<td>WE8EBCDIC1146</td>
<td>EBCDIC Code Page 1146 8-bit West European</td>
<td>9F</td>
</tr>
<tr>
<td>WE8EBCDIC1148</td>
<td>EBCDIC Code Page 1148 8-bit West European</td>
<td>9F</td>
</tr>
<tr>
<td>WE8EBCDIC1148C</td>
<td>EBCDIC Code Page 1148 Client 8-bit West European</td>
<td>9F</td>
</tr>
<tr>
<td>EL8ISO8859P7</td>
<td>ISO 8859-7 Latin/Greek</td>
<td>A4</td>
</tr>
<tr>
<td>IW8MSWIN1255</td>
<td>MS Windows Code Page 1255 8-bit Latin/Hebrew</td>
<td>80</td>
</tr>
<tr>
<td>AR8MSWIN1256</td>
<td>MS Windows Code Page 1256 8-Bit Latin/Arabic</td>
<td>80</td>
</tr>
<tr>
<td>TR8MSWIN1254</td>
<td>MS Windows Code Page 1254 8-bit Turkish</td>
<td>80</td>
</tr>
<tr>
<td>BLT8MSWIN1257</td>
<td>MS Windows Code Page 1257 Baltic</td>
<td>80</td>
</tr>
<tr>
<td>VN8MSWIN1258</td>
<td>MS Windows Code Page 1258 8-bit Vietnamese</td>
<td>80</td>
</tr>
<tr>
<td>TH8TISASCII</td>
<td>Thai Industrial 620-2533 - ASCII 8-bit</td>
<td>80</td>
</tr>
<tr>
<td>AL32UTF8</td>
<td>Unicode 3.2 UTF-8 Universal character set</td>
<td>E282AC</td>
</tr>
<tr>
<td>UTF8</td>
<td>CESU-8</td>
<td>E282AC</td>
</tr>
<tr>
<td>AL16UTF16</td>
<td>Unicode 3.2 UTF-16 Universal character set</td>
<td>20AC</td>
</tr>
<tr>
<td>UTFE</td>
<td>UTF-EBCDIC encoding of Unicode 3.0</td>
<td>CA4653</td>
</tr>
<tr>
<td>ZHT16HKSCS</td>
<td>MS Windows Code Page 950 with Hong Kong Supp Character Set</td>
<td></td>
</tr>
<tr>
<td>ZHS32GB18030</td>
<td>GB18030-2000</td>
<td>A2E3</td>
</tr>
<tr>
<td>WE8BS2000E</td>
<td>Siemens EBCDIC.DE.04 8-bit West European</td>
<td>9F</td>
</tr>
</tbody>
</table>
Oracle Support for the Euro

Twelve members of the European Monetary Union (EMU) have adopted the euro as their currency. Setting NLS_TERRITORY to correspond to a country in the EMU (Austria, Belgium, Finland, France, Germany, Greece, Ireland, Italy, Luxembourg, the Netherlands, Portugal, and Spain) results in the default values for NLS_CURRENCY and NLS_DUAL_CURRENCY being set to EUR.

During the transition period (1999 through 2001), Oracle support for the euro was provided in Oracle8i and later as follows:

- NLS_CURRENCY was defined as the primary currency of the country
- NLS_ISO_CURRENCY was defined as the ISO currency code of a given territory
- NLS_DUAL_CURRENCY was defined as the secondary currency symbol (usually the euro) for a given territory

Beginning with Oracle9i release 2 (9.2), the value of NLS_ISO_CURRENCY results in the ISO currency symbol being set to EUR for EMU member countries who use the euro. For example, suppose NLS_ISO_CURRENCY is set to FRANCE. Enter the following SELECT statement:

```
SELECT TO_CHAR(TOTAL, 'C099G999D99') "TOTAL" FROM orders WHERE customer_id=585;
```

You should see output similar to the following:

```
TOTAL
-------
EUR12.673,49
```

Customers who must retain their obsolete local currency symbol can override the default for NLS_DUAL_CURRENCY or NLS_CURRENCY by defining them as parameters in the initialization file on the server and as environment variables on the client.

---

**Note:** NLS_LANG must also be set on the client for NLS_CURRENCY or NLS_DUAL_CURRENCY to take effect.

---

It is not possible to override the ISO currency symbol that results from the value of NLS_ISO_CURRENCY.
Monetary Parameters

NLS_MONETARY_CHARACTERS

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter scope</td>
<td>Environment variable</td>
</tr>
<tr>
<td>Default value</td>
<td>Derived from NLS_TERRITORY</td>
</tr>
<tr>
<td>Range of values</td>
<td>Any valid character</td>
</tr>
</tbody>
</table>

NLS_MONETARY_CHARACTERS specifies the character that separates groups of numbers in monetary expressions. For example, when the territory is America, the thousands separator is a comma, and the decimal separator is a period.

NLS_CREDIT

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter scope</td>
<td>Environment variable</td>
</tr>
<tr>
<td>Default value</td>
<td>Derived from NLS_TERRITORY</td>
</tr>
<tr>
<td>Range of values</td>
<td>Any string, maximum of 9 bytes (not including null)</td>
</tr>
</tbody>
</table>

NLS_CREDIT sets the symbol that displays a credit in financial reports. The default value of this parameter is determined by NLS_TERRITORY. For example, a space is a valid value of NLS_CREDIT.

This parameter can be specified only in the client environment.

It can be retrieved through the OCIGetNlsInfo() function.

NLS_DEBIT

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter scope</td>
<td>Environment variable</td>
</tr>
<tr>
<td>Default value</td>
<td>Derived from NLS_TERRITORY</td>
</tr>
</tbody>
</table>
NLS_DEBIT sets the symbol that displays a debit in financial reports. The default value of this parameter is determined by NLS_TERRITORY. For example, a minus sign (-) is a valid value of NLS_DEBIT.

This parameter can be specified only in the client environment. It can be retrieved through the `OCIGetNlsInfo()` function.

**Linguistic Sort Parameters**

You can choose how to sort data by using linguistic sort parameters.

This section includes the following topics:

- **NLS_SORT**
- **NLS_COMP**

**See Also:** Chapter 5, "Linguistic Sorting and String Searching"

**NLS_SORT**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter scope</td>
<td>Initialization parameter, environment variable, ALTER SESSION, and SQL functions</td>
</tr>
<tr>
<td>Default value</td>
<td>Derived from NLS_LANGUAGE</td>
</tr>
<tr>
<td>Range of values</td>
<td>BINARY or any valid linguistic sort name</td>
</tr>
</tbody>
</table>

NLS_SORT specifies the type of sort for character data. It overrides the default value that is derived from NLS_LANGUAGE.

The syntax of NLS_SORT is:

```
NLS_SORT = BINARY | sort_name
```

BINARY specifies a binary sort. `sort_name` specifies a linguistic sort sequence.
The value of NLS_SORT affects the following SQL operations: WHERE, START WITH, IN/OUT, BETWEEN, CASE WHEN, HAVING, ORDER BY. All other SQL operators make comparisons in binary mode only.

Example 3–21 Setting NLS_SORT
To specify the German linguistic sort sequence, set NLS_SORT as follows:

NLS_SORT = German

---

Note: When the NLS_SORT parameter is set to BINARY, the optimizer can, in some cases, satisfy the ORDER BY clause without doing a sort by choosing an index scan.

When NLS_SORT is set to a linguistic sort, a sort is needed to satisfy the ORDER BY clause if there is no linguistic index for the linguistic sort specified by NLS_SORT.

If a linguistic index exists for the linguistic sort specified by NLS_SORT, then the optimizer can, in some cases, satisfy the ORDER BY clause without doing a sort by choosing an index scan.

---

You can alter the default value of NLS_SORT by doing one of the following:

- Changing its value in the initialization parameter file and then restarting the instance
- Using an ALTER SESSION statement

See Also:

- Chapter 5, "Linguistic Sorting and String Searching"
- Oracle Database SQL Reference for more information about the ALTER SESSION statement
- "Linguistic Sorts" on page A-25 for a list of linguistic sort names

### NLS_COMP

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
</tbody>
</table>
The value of NLS_COMP affects the following SQL operations: WHERE, START WITH, IN/OUT, BETWEEN, CASE WHEN, HAVING, ORDER BY. All other SQL operators make comparisons in binary mode only.

You can use NLS_COMP to avoid the cumbersome process of using the NLSSORT function in SQL statements when you want to perform a linguistic comparison instead of a binary comparison. When NLS_COMP is set to ANSI, SQL operations perform a linguistic comparison based on the value of NLS_SORT.

Set NLS_COMP to ANSI as follows:

```
ALTER SESSION SET NLS_COMP = ANSI;
```

When NLS_COMP is set to ANSI, a linguistic index improves the performance of the linguistic comparison.

To enable a linguistic index, use the following syntax:

```
CREATE INDEX i ON t(NLSSORT(col, 'NLS_SORT=FRENCH'));
```

See Also:

- "Using Linguistic Sorts" on page 5-3
- "Using Linguistic Indexes" on page 5-19

## Character Set Conversion Parameter

This section includes the following topic:

- **NLS_NCHAR_CONV_EXCP**

## NLS_NCHAR_CONV_EXCP

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
</tbody>
</table>

**Property Description**

<table>
<thead>
<tr>
<th>Property scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization parameter, environment variable, and ALTER SESSION</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Default value</th>
<th>BINARY</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Range of values</th>
<th>BINARY or ANSI</th>
</tr>
</thead>
</table>

Setting Up a Globalization Support Environment  3-43
NLS_NCHAR_CONV_EXCP determines whether an error is reported when there is data loss during an implicit or explicit character type conversion between NCHAR/NVARCHAR data and CHAR/VARCHAR2 data. The default value results in no error being reported.

See Also: Chapter 11, "Character Set Migration" for more information about data loss during character set conversion

Length Semantics

This section includes the following topic:

- **NLS_LENGTH_SEMANTICS**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter type</td>
<td>String</td>
</tr>
<tr>
<td>Parameter scope</td>
<td>Environment variable, initialization parameter, ALTER SESSION, and ALTER SYSTEM</td>
</tr>
<tr>
<td>Default value</td>
<td>BYTE</td>
</tr>
<tr>
<td>Range of values</td>
<td>BYTE or CHAR</td>
</tr>
</tbody>
</table>

By default, the character datatypes CHAR and VARCHAR2 are specified in bytes, not characters. Hence, the specification CHAR(20) in a table definition allows 20 bytes for storing character data.

This works well if the database character set uses a single-byte character encoding scheme because the number of characters is the same as the number of bytes. If the database character set uses a multibyte character encoding scheme, then the number of bytes no longer equals the number of characters because a character can consist of one or more bytes. Thus, column widths must be chosen with care to
allow for the maximum possible number of bytes for a given number of characters. You can overcome this problem by switching to character semantics when defining the column size.

NLS_LENGTH_SEMANTICS enables you to create CHAR, VARCHAR2, and LONG columns using either byte or character length semantics. NCHAR, NVARCHAR2, CLOB, and NCLOB columns are always character-based. Existing columns are not affected. You may be required to use byte semantics in order to maintain compatibility with existing applications.

NLS_LENGTH_SEMANTICS does not apply to tables in SYS and SYSTEM. The data dictionary always uses byte semantics.

Note that if the NLS_LENGTH_SEMANTICS environment variable is not set on the client, then the client session defaults to the value for NLS_LENGTH_SEMANTICS on the database server. This enables all client sessions on the network to have the same NLS_LENGTH_SEMANTICS behavior. Setting the environment variable on an individual client enables the server initialization parameter to be overridden for that client.

See Also:

- "Length Semantics" on page 2-12
- Oracle Database Concepts for more information about length semantics
This chapter includes the following topics:

- Overview of Datetime and Interval Datatypes and Time Zone Support
- Datetime and Interval Datatypes
- Datetime and Interval Arithmetic and Comparisons
- Datetime SQL Functions
- Datetime and Time Zone Parameters and Environment Variables
- Choosing a Time Zone File
- Setting the Database Time Zone
- Converting Time Zones With the AT TIME ZONE Clause
- Setting the Session Time Zone
- Support for Daylight Saving Time
Overview of Datetime and Interval Datatypes and Time Zone Support

Businesses conduct transactions across time zones. Oracle’s datetime and interval datatypes and time zone support make it possible to store consistent information about the time of events and transactions.

Note: This chapter describes Oracle datetime and interval datatypes. It does not attempt to describe ANSI datatypes or other kinds of datatypes except when noted.

Datetime and Interval Datatypes

The datetime datatypes are DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, and TIMESTAMP WITH LOCAL TIME ZONE. Values of datetime datatypes are sometimes called datetimes.

The interval datatypes are INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND. Values of interval datatypes are sometimes called intervals.

Both datetimes and intervals are made up of fields. The values of these fields determine the value of the datatype. The fields that apply to all Oracle datetime and interval datatypes are:

- YEAR
- MONTH
- DAY
- HOUR
- MINUTE
- SECOND

TIMESTAMP WITH TIME ZONE also includes these fields:

- TIMEZONE_HOUR
- TIMEZONE_MINUTE
- TIMEZONE_REGION
- TIMEZONE_ABBR
TIMESTAMP WITH LOCAL TIME ZONE does not store time zone information, but you can see local time zone information in output if the `TZH:TZM` or `TZR, TZD` format elements are specified.

The following sections describe the datetime datatypes and interval datatypes in more detail:

- Datetime Datatypes
- Interval Datatypes

**See Also:** Oracle Database SQL Reference for the valid values of the datetime and interval fields. Oracle Database SQL Reference also contains information about format elements.

### Datetime Datatypes

This section includes the following topics:

- DATE Datatype
- TIMESTAMP Datatype
- TIMESTAMP WITH TIME ZONE Datatype
- TIMESTAMP WITH LOCAL TIME ZONE Datatype
- Inserting Values into Datetime Datatypes
- Choosing a TIMESTAMP Datatype

#### DATE Datatype

The DATE datatype stores date and time information. Although date and time information can be represented in both character and number datatypes, the DATE datatype has special associated properties. For each DATE value, Oracle stores the following information: century, year, month, date, hour, minute, and second.

You can specify a date value by:

- Specifying the date value as a literal
- Converting a character or numeric value to a date value with the TO_DATE function

A date can be specified as an ANSI date literal or as an Oracle date value.

An ANSI date literal contains no time portion and must be specified in exactly the following format:
DATE 'YYYY-MM-DD'

The following is an example of an ANSI date literal:
DATE '1998-12-25'

Alternatively, you can specify an Oracle date value as shown in the following example:

```
TO_DATE('1998-DEC-25 17:30','YYYY-MON-DD HH24:MI','NLS_DATE_LANGUAGE=AMERICAN')
```

The default date format for an Oracle date value is derived from the NLS_DATE_FORMAT and NLS_DATE_LANGUAGE initialization parameters. The date format in the example includes a two-digit number for the day of the month, an abbreviation of the month name, the last two digits of the year, and a 24-hour time designation. The specification for NLS_DATE_LANGUAGE is included because ‘DEC’ is not a valid value for MON in all locales.

Oracle automatically converts character values that are in the default date format into date values when they are used in date expressions.

If you specify a date value without a time component, then the default time is midnight. If you specify a date value without a date, then the default date is the first day of the current month.

Oracle DATE columns always contain fields for both date and time. If your queries use a date format without a time portion, then you must ensure that the time fields in the DATE column are set to midnight. You can use the TRUNC (date) SQL function to ensure that the time fields are set to midnight, or you can make the query a test of greater than or less than (<?, <=, >=, or >) instead of equality or inequality (= or !=). Otherwise, Oracle may not return the query results you expect.

**See Also:**
- Oracle Database SQL Reference for more information about the DATE datatype
- "NLS_DATE_FORMAT" on page 3-21
- "NLS_DATE_LANGUAGE" on page 3-22
- Oracle Database SQL Reference for more information about literals, format elements such as MM, and the TO_DATE function
TIMESTAMP Datatype

The TIMESTAMP datatype is an extension of the DATE datatype. It stores year, month, day, hour, minute, and second values. It also stores fractional seconds, which are not stored by the DATE datatype.

Specify the TIMESTAMP datatype as follows:

```
TIMESTAMP [(fractional_seconds_precision)]
```

`fractional_seconds_precision` is optional and specifies the number of digits in the fractional part of the SECOND datetime field. It can be a number in the range 0 to 9. The default is 6.

For example, '26-JUN-02 09:39:16.78' shows 16.78 seconds. The fractional seconds precision is 2 because there are 2 digits in '78'.

You can specify the TIMESTAMP literal in a format like the following:

```
TIMESTAMP 'YYYY-MM-DD HH24:MI:SS.FF'
```

Using the example format, specify TIMESTAMP as a literal as follows:

```
TIMESTAMP '1997-01-31 09:26:50.12'
```

The value of NLS_TIMESTAMP_FORMAT initialization parameter determines the timestamp format when a character string is converted to the TIMESTAMP datatype. NLS_DATE_LANGUAGE determines the language used for character data such as MON.

See Also:

- Oracle Database SQL Reference for more information about the TIMESTAMP datatype
- "NLS_TIMESTAMP_FORMAT" on page 3-24
- "NLS_DATE_LANGUAGE" on page 3-22

TIMESTAMP WITH TIME ZONE Datatype

TIMESTAMP WITH TIME ZONE is a variant of TIMESTAMP that includes a time zone offset or time zone region name in its value. The time zone offset is the difference (in hours and minutes) between local time and UTC (Coordinated Universal Time, formerly Greenwich Mean Time). Specify the TIMESTAMP WITH TIME ZONE datatype as follows:

```
TIMESTAMP [((fractional_seconds_precision)) WITH TIME ZONE
```
fractional_seconds_precision is optional and specifies the number of digits in the fractional part of the SECOND datetime field.

You can specify `TIMESTAMP WITH TIME ZONE` as a literal as follows:

```
TIMESTAMP '1997-01-31 09:26:56.66 +02:00'
```

Two `TIMESTAMP WITH TIME ZONE` values are considered identical if they represent the same instant in UTC, regardless of the `TIME ZONE` offsets stored in the data. For example, the following expressions have the same value:

```
TIMESTAMP '1999-01-15 8:00:00 -8:00'
TIMESTAMP '1999-01-15 11:00:00 -5:00'
```

You can replace the UTC offset with the TZR (time zone region) format element. The following expression specifies US/Pacific for the time zone region:

``````TIMESTAMP '1999-01-15 8:00:00 US/Pacific'
```

To eliminate the ambiguity of boundary cases when the time switches from Standard Time to Daylight Saving Time, use both the TZR format element and the corresponding TZD format element. The TZD format element is an abbreviation of the time zone region with Daylight Saving Time information included. Examples are PST for US/Pacific standard time and PDT for US/Pacific daylight time. The following specification ensures that a Daylight Saving Time value is returned:

```
TIMESTAMP '1999-10-29 01:30:00 US/Pacific PDT'
```

If you do not add the TZD format element, and the datetime value is ambiguous, then Oracle returns an error if you have the `ERROR_ON_OVERLAP_TIME` session parameter set to TRUE. If `ERROR_ON_OVERLAP_TIME` is set to FALSE (the default value), then Oracle interprets the ambiguous datetime as Standard Time.

The default date format for the `TIMESTAMP WITH TIME ZONE` datatype is determined by the value of the `NLS_TIMESTAMP_TZ_FORMAT` initialization parameter.
TIMESTAMP WITH LOCAL TIME ZONE Datatype

TIMESTAMP WITH LOCAL TIME ZONE is another variant of TIMESTAMP. It differs from TIMESTAMP WITH TIME ZONE as follows: data stored in the database is normalized to the database time zone, and the time zone offset is not stored as part of the column data. When users retrieve the data, Oracle returns it in the users’ local session time zone. The time zone offset is the difference (in hours and minutes) between local time and UTC (Coordinated Universal Time, formerly Greenwich Mean Time).

Specify the TIMESTAMP WITH LOCAL TIME ZONE datatype as follows:

```
TIMESTAMP [(fractional_seconds_precision)] WITH LOCAL TIME ZONE
```

`fractional_seconds_precision` is optional and specifies the number of digits in the fractional part of the SECOND datetime field.

There is no literal for TIMESTAMP WITH LOCAL TIME ZONE, but TIMESTAMP literals and TIMESTAMP WITH TIME ZONE literals can be inserted into a TIMESTAMP WITH LOCAL TIME ZONE column.

The default date format for TIMESTAMP WITH LOCAL TIME ZONE is determined by the value of the NLS_TIMESTAMP_FORMAT initialization parameter.
Inserting Values into Datetime Datatypes

You can insert values into a datetime column in the following ways:

- Insert a character string whose format is based on the appropriate NLS format value
- Insert a literal
- Insert a literal for which implicit conversion is performed
- Use the `TO_TIMESTAMP`, `TO_TIMESTAMP_TZ`, or `TO_DATE` SQL function

The following examples show how to insert data into datetime datatypes.

**Example 4–1 Inserting Data into a DATE Column**

Set the date format.

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT='DD-MON-YYYY HH24:MI:SS';
```

Create a table `table_dt` with columns `c_id` and `c_dt`. The `c_id` column is of `NUMBER` datatype and helps to identify the method by which the data is entered. The `c_dt` column is of `DATE` datatype.

```
SQL> CREATE TABLE table_dt (c_id NUMBER, c_dt DATE);
```

Insert a date as a character string.

```
SQL> INSERT INTO table_dt VALUES(1, '01-JAN-2003');
```

Insert the same date as a `DATE` literal.

```
SQL> INSERT INTO table_dt VALUES(2, DATE '2003-01-01');
```

Insert the date as a `TIMESTAMP` literal. Oracle drops the time zone information.

```
SQL> INSERT INTO table_dt VALUES(3, TIMESTAMP '2003-01-01 00:00:00 US/Pacific');
```

See Also:

- *Oracle Database SQL Reference* for more information about the `TIMESTAMP WITH LOCAL TIME ZONE` datatype
- "TIMESTAMP Datatype" on page 4-5 for more information about fractional seconds precision
- "NLS_TIMESTAMP_FORMAT" on page 3-24
Insert the date with the `TO_DATE` function.

```
SQL> INSERT INTO table_dt VALUES(4, TO_DATE('01-JAN-2003', 'DD-MON-YYYY'));
```

Display the data.

```
SQL> SELECT * FROM table_dt;
```

```
C_ID  C_DT
------ --------------------
1      01-JAN-2003 00:00:00
2      01-JAN-2003 00:00:00
3      01-JAN-2003 00:00:00
4      01-JAN-2003 00:00:00
```

**Example 4-2 Inserting Data into a TIMESTAMP Column**

Set the timestamp format.

```
SQL> ALTER SESSION SET NLS_TIMESTAMP_FORMAT='DD-MON-YY HH:MI:SSF';
```

Create a table `table_ts` with columns `c_id` and `c_ts`. The `c_id` column is of `NUMBER` datatype and helps to identify the method by which the data is entered. The `c_ts` column is of `TIMESTAMP` datatype.

```
SQL> CREATE TABLE table_ts(c_id NUMBER, c_ts TIMESTAMP);
```

Insert a date and time as a character string.

```
SQL> INSERT INTO table_ts VALUES(1, '01-JAN-2003 2:00:00');
```

Insert the same date and time as a `TIMESTAMP` literal.

```
SQL> INSERT INTO table_ts VALUES(2, TIMESTAMP '2003-01-01 2:00:00');
```

Insert the same date and time as a `TIMESTAMP WITH TIME ZONE` literal. Oracle converts it to a `TIMESTAMP` value, which means that the time zone information is dropped.

```
SQL> INSERT INTO table_ts VALUES(3, TIMESTAMP '2003-01-01 2:00:00 -08:00');
```

Display the data.

```
SQL> SELECT * FROM table_ts;
```

```
C_ID  C_TS
------ -----------------------------
1      01-JAN-03 02:00:00.000000 AM
2      01-JAN-03 02:00:00.000000 AM
```
Note that the three methods result in the same value being stored.

**Example 4-3  Inserting Data into the TIMESTAMP WITH TIME ZONE Datatype**

Set the timestamp format.

```sql
SQL> ALTER SESSION SET NLS_TIMESTAMP__TZ_FORMAT='DD-MON-RR HH:MI:SSFF AM TZR';
```

Set the time zone to ‘-07:00’.

```sql
SQL> ALTER SESSION SET TIME_ZONE='-7:00';
```

Create a table `table_tstz` with columns `c_id` and `c_tstz`. The `c_id` column is of `NUMBER` datatype and helps to identify the method by which the data is entered. The `c_tstz` column is of `TIMESTAMP WITH TIME ZONE` datatype.

```sql
SQL> CREATE TABLE table_tstz (c_id NUMBER, c_tstz TIMESTAMP WITH TIME ZONE);
```

Insert a date and time as a character string.

```sql
SQL> INSERT INTO table_tstz VALUES(1, '01-JAN-2003 2:00:00 AM -07:00');
```

Insert the same date and time as a `TIMESTAMP` literal. Oracle converts it to a `TIMESTAMP WITH TIME ZONE` literal, which means that the session time zone is appended to the `TIMESTAMP` value.

```sql
SQL> INSERT INTO table_tstz VALUES(2, TIMESTAMP '2003-01-01 2:00:00');
```

Insert the same date and time as a `TIMESTAMP WITH TIME ZONE` literal.

```sql
SQL> INSERT INTO table_tstz VALUES(3, TIMESTAMP '2003-01-01 2:00:00 -8:00');
```

Display the data.

```sql
SQL> SELECT * FROM table_tstz;
C_ID  C_TSTZ
------- ------------------------------------
1  01-JAN-03 02:00:00.000000 AM -07:00
2  01-JAN-03 02:00:00.000000 AM -07:00
3  01-JAN-03 02:00:00.000000 AM -08:00
```

Note that the time zone is different for method 3, because the time zone information was specified as part of the `TIMESTAMP WITH TIME ZONE` literal.
Example 4–4  Inserting Data into the `TIMESTAMP WITH LOCAL TIME ZONE` Datatype

Consider data that is being entered in Denver, Colorado, U.S.A., whose time zone is UTC-7.

SQL> ALTER SESSION SET TIME_ZONE='07:00';

Create a table `table_tsltz` with columns `c_id` and `c_tsltz`. The `c_id` column is of `NUMBER` datatype and helps to identify the method by which the data is entered. The `c_tsltz` column is of `TIMESTAMP WITH LOCAL TIME ZONE` datatype.

SQL> CREATE TABLE table_tsltz (c_id NUMBER, c_tsltz TIMESTAMP WITH LOCAL TIME ZONE);

Insert a date and time as a character string.

SQL> INSERT INTO table_tsltz VALUES(1, '01-JAN-2003 2:00:00');

Insert the same data as a `TIMESTAMP WITH LOCAL TIME ZONE` literal.

SQL> INSERT INTO table_tsltz VALUE(2, TIMESTAMP '2003-01-01 2:00:00');

Insert the same data as a `TIMESTAMP WITH TIME ZONE` literal. Oracle converts the data to a `TIMESTAMP WITH LOCAL TIME ZONE` value. This means the time zone that is entered (`-08:00`) is converted to the session time zone value (`-07:00`).

SQL> INSERT INTO table_tsltz VALUES(3, TIMESTAMP '2003-01-01 2:00:00 -08:00');

Display the data.

SQL> SELECT * FROM table_tsltz;

```
   C_ID   C_TSLTZ
---------- ------------------------------------
   1 01-JAN-03 02.00.00.000000 AM
   2 01-JAN-03 02.00.00.000000 AM
   3 01-JAN-03 03.00.00.000000 AM
```

Note that the information that was entered as UTC-8 has been changed to the local time zone, changing the hour from 2 to 3.

See Also:  "Datetime SQL Functions" on page 4-16 for more information about the `TO_TIMESTAMP` or `TO_TIMESTAMP_TZ` SQL functions
Choosing a TIMESTAMP Datatype

Use the `TIMESTAMP` datatype when you need a datetime value without locale information. For example, you can store information about the times when workers punch a timecard in and out of their assembly line workstations. The `TIMESTAMP` datatype uses 7 or 11 bytes of storage.

Use the `TIMESTAMP WITH TIME ZONE` datatype when the application is used across time zones. Consider a banking company with offices around the world. It records a deposit to an account at 11 a.m. in London and a withdrawal of the same amount from the account at 9 a.m. in New York. The money is in the account for four hours. Unless time zone information is stored with the account transactions, it appears that the account is overdrawn from 9 a.m. to 11 a.m.

The `TIMESTAMP WITH TIME ZONE` datatype requires 13 bytes of storage, or two more bytes of storage than the `TIMESTAMP` and `TIMESTAMP WITH LOCAL TIME ZONE` datatypes because it stores time zone information. The time zone is stored as an offset from UTC or as a time zone region name. The data is available for display or calculations without additional processing. A `TIMESTAMP WITH TIME ZONE` column cannot be used as a primary key. If an index is created on a `TIMESTAMP WITH TIME ZONE` column, it becomes a function-based index.

The `TIMESTAMP WITH LOCAL TIME ZONE` datatype stores the timestamp without time zone information. It normalizes the data to the database time zone every time the data is sent to and from a client. It requires 11 bytes of storage.

The `TIMESTAMP WITH LOCAL TIME ZONE` datatype is appropriate when the original time zone is of no interest, but the relative times of events are important. Consider the transactions described in the previous banking example. Suppose the data is recorded using the `TIMESTAMP WITH LOCAL TIME ZONE` datatype. If the database time zone of the bank is set to `Asia/Hong_Kong`, then an employee in Hong Kong who displays the data would see that the deposit was made at 1900 and the withdrawal was made at 2300. If the same data is displayed in London, it would show that the deposit was made at 1100 and the withdrawal was made at 1500. The four-hour difference is preserved, but the actual times are not, making it impossible to tell whether the transactions were done during business hours.

Interval Datatypes

Interval datatypes store time durations. They are used primarily with analytic functions. For example, you can use them to calculate a moving average of stock prices. You must use interval datatypes to determine the values that correspond to a particular percentile. You can also use interval datatypes to update historical tables.
This section includes the following topics:

- INTERVAL YEAR TO MONTH Datatype
- INTERVAL DAY TO SECOND Datatype
- Inserting Values into Interval Datatypes

**See Also:** *Oracle Data Warehousing Guide* for more information about analytic functions, including moving averages (and inverse percentiles)

### INTERVAL YEAR TO MONTH Datatype

INTERVAL YEAR TO MONTH stores a period of time using the YEAR and MONTH datetime fields. Specify INTERVAL YEAR TO MONTH as follows:

```
INTERVAL YEAR [(year_precision)] TO MONTH
```

*year_precision* is the number of digits in the YEAR datetime field. Accepted values are 0 to 9. The default value of *year_precision* is 2.

Interval values can be specified as literals. There are many ways to specify interval literals. The following is one example of specifying an interval of 123 years and 2 months. The year precision is 3.

```
INTERVAL '123-2' YEAR(3) TO MONTH
```

**See Also:** *Oracle Database SQL Reference* for more information about specifying interval literals with the INTERVAL YEAR TO MONTH datatype

### INTERVAL DAY TO SECOND Datatype

INTERVAL DAY TO SECOND stores a period of time in terms of days, hours, minutes, and seconds. Specify this datatype as follows:

```
INTERVAL DAY [(day_precision)] TO SECOND [(fractional_seconds_precision)]
```

*day_precision* is the number of digits in the DAY datetime field. Accepted values are 0 to 9. The default is 2.

*fractional_seconds_precision* is the number of digits in the fractional part of the SECOND datetime field. Accepted values are 0 to 9. The default is 6.
The following is one example of specifying an interval of 4 days, 5 hours, 12 minutes, 10 seconds, and 222 thousandths of a second. The fractional second precision is 3.

\[ \text{INTERVAL '4 5:12:10.222' DAY TO SECOND(3)} \]

Interval values can be specified as literals. There are many ways to specify interval literals.

**See Also:** Oracle Database SQL Reference for more information about specifying interval literals with the INTERVAL DAY TO SECOND datatype

### Inserting Values into Interval Datatypes

You can insert values into an interval column in the following ways:

- Insert an interval as a literal. For example:

  ```sql
  INSERT INTO table1 VALUES (INTERVAL '4-2' YEAR TO MONTH);
  ```

  This statement inserts an interval of 4 years and 2 months.

- Oracle recognizes literals for other ANSI interval types and converts the values to Oracle interval values.

- Use the `NUMTODSINTERVAL`, `NUMTOYMINTERVAL`, `TO_DSINTERVAL`, and `TO_YMINTERVAL` SQL functions.

  **See Also:** "Datetime SQL Functions" on page 4-16

### Datetime and Interval Arithmetic and Comparisons

This section includes the following topics:

- Datetime and Interval Arithmetic
- Datetime Comparisons
- Explicit Conversion of Datetime Datatypes

### Datetime and Interval Arithmetic

You can perform arithmetic operations on date (DATE), timestamp (TIMESTAMP, TIMESTAMP WITH TIME ZONE, and TIMESTAMP WITH LOCAL TIME ZONE) and interval (INTERVAL DAY TO SECOND and INTERVAL YEAR TO MONTH) data.
You can maintain the most precision in arithmetic operations by using a timestamp datatype with an interval datatype.

You can use NUMBER constants in arithmetic operations on date and timestamp values. Oracle internally converts timestamp values to date values before doing arithmetic operations on them with NUMBER constants. This means that information about fractional seconds is lost during operations that include both date and timestamp values. Oracle interprets NUMBER constants in datetime and interval expressions as number of days.

Each DATE value contains a time component. The result of many date operations includes a fraction. This fraction means a portion of one day. For example, 1.5 days is 36 hours. These fractions are also returned by Oracle built-in SQL functions for common operations on DATE data. For example, the built-in MONTHS_BETWEEN SQL function returns the number of months between two dates. The fractional portion of the result represents that portion of a 31-day month.

Oracle performs all timestamp arithmetic in UTC time. For TIMESTAMP WITH LOCAL TIME ZONE data, Oracle converts the datetime value from the database time zone to UTC and converts back to the database time zone after performing the arithmetic. For TIMESTAMP WITH TIME ZONE data, the datetime value is always in UTC, so no conversion is necessary.

See Also:

- Oracle Database SQL Reference for detailed information about datetime and interval arithmetic operations
- "Datetime SQL Functions" on page 4-16 for information about which functions cause implicit conversion to DATE

Datetime Comparisons

When you compare date and timestamp values, Oracle converts the data to the more precise datatype before doing the comparison. For example, if you compare data of TIMESTAMP WITH TIME ZONE datatype with data of TIMESTAMP datatype, Oracle converts the TIMESTAMP data to TIMESTAMP WITH TIME ZONE, using the session time zone.

The order of precedence for converting date and timestamp data is as follows:

1. DATE
2. TIMESTAMP
3. TIMESTAMP WITH LOCAL TIME ZONE
4. **TIMESTAMP WITH TIME ZONE**

For any pair of datatypes, Oracle converts the datatype that has a smaller number in the preceding list to the datatype with the larger number.

**Explicit Conversion of Datetime Datatypes**

If you want to do explicit conversion of datetime datatypes, use the `CAST` SQL function. You can explicitly convert `DATE`, `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `TIMESTAMP WITH LOCAL TIME ZONE` to another datatype in the list.

**See Also:** *Oracle Database SQL Reference*

**Datetime SQL Functions**

Datetime functions operate on date (`DATE`), timestamp (`TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `TIMESTAMP WITH LOCAL TIME ZONE`) and interval (`INTERVAL DAY TO SECOND`, `INTERVAL YEAR TO MONTH`) values.

Some of the datetime functions were designed for the Oracle `DATE` datatype. If you provide a timestamp value as their argument, then Oracle internally converts the input type to a `DATE` value. Oracle does not perform internal conversion for the `ROUND` and `TRUNC` functions.

*Table 4–1* shows the datetime functions that were designed for the Oracle `DATE` datatype. For more detailed descriptions, refer to *Oracle Database SQL Reference*.

**Table 4–1 Datetime Functions Designed for the `DATE` Datatype**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ADD_MONTHS</code></td>
<td>Returns the date ( d ) plus ( n ) months</td>
</tr>
<tr>
<td><code>LAST_DAY</code></td>
<td>Returns the last day of the month that contains ( date )</td>
</tr>
<tr>
<td><code>MONTHS_BETWEEN</code></td>
<td>Returns the number of months between ( date1 ) and ( date2 )</td>
</tr>
<tr>
<td><code>NEW_TIME</code></td>
<td>Returns the date and time in <code>zone2</code> time zone when the date and time in <code>zone1</code> time zone are <code>date</code></td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>This function takes as input only a limited number of time zones. You can have access to a much greater number of time zones by combining the <code>FROM_TZ</code> function and the datetime expression.</td>
</tr>
<tr>
<td><code>NEXT_DAY</code></td>
<td>Returns the date of the first weekday named by <code>char</code> that is later than <code>date</code></td>
</tr>
</tbody>
</table>

4-16 Oracle Database Globalization Support Guide
Table 4–2 describes additional datetime functions. For more detailed descriptions, refer to Oracle Database SQL Reference.

### Table 4–2 Additional Datetime Functions

<table>
<thead>
<tr>
<th>Datetime Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURRENT_DATE</td>
<td>Returns the current date in the session time zone in a value in the Gregorian calendar, of the DATE datatype</td>
</tr>
<tr>
<td>CURRENT_TIMESTAMP</td>
<td>Returns the current date and time in the session time zone as a TIMESTAMP WITH TIME ZONE value</td>
</tr>
<tr>
<td>DBTIMEZONE</td>
<td>Returns the value of the database time zone. The value is a time zone offset or a time zone region name.</td>
</tr>
<tr>
<td>EXTRACT (datetime)</td>
<td>Extracts and returns the value of a specified datetime field from a datetime or interval value expression</td>
</tr>
<tr>
<td>FROM_TZ</td>
<td>Converts a TIMESTAMP value at a time zone to a TIMESTAMP WITH TIME ZONE value</td>
</tr>
<tr>
<td>LOCALTIMESTAMP</td>
<td>Returns the current date and time in the session time zone in a value of the TIMESTAMP datatype</td>
</tr>
<tr>
<td>NUMTODSINTERVAL</td>
<td>Converts number ( n ) to an INTERVAL DAY TO SECOND literal</td>
</tr>
<tr>
<td>NUMTOYMINTERVAL</td>
<td>Converts number ( n ) to an INTERVAL YEAR TO MONTH literal</td>
</tr>
<tr>
<td>SESSIONTIMEZONE</td>
<td>Returns the value of the current session’s time zone</td>
</tr>
<tr>
<td>SYS_EXTRACT_UTC</td>
<td>Extracts the UTC from a datetime with time zone offset</td>
</tr>
<tr>
<td>SYSDATE</td>
<td>Returns the date and time of the operating system on which the database resides, taking into account the time zone of the database server’s operating system that was in effect when the database was started.</td>
</tr>
<tr>
<td>SYSTIMESTAMP</td>
<td>Returns the system date, including fractional seconds and time zone of the system on which the database resides</td>
</tr>
</tbody>
</table>
### Datetime and Time Zone Parameters and Environment Variables

This section includes the following topics:

- Datetime Format Parameters
- Time Zone Environment Variables
- Daylight Saving Time Session Parameter

#### Datetime Format Parameters

Table 4–3 contains the names and descriptions of the datetime format parameters.

**Table 4–2 Additional Datetime Functions (Cont.)**

<table>
<thead>
<tr>
<th>Datetime Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TO_CHAR</code> (datetime)</td>
<td>Converts a datetime or interval value of <code>DATE</code>, <code>TIMESTAMP</code>, <code>TIMESTAMP WITH TIME ZONE</code>, or <code>TIMESTAMP WITH LOCAL TIME ZONE</code> datatype to a value of <code>VARCHAR2</code> datatype in the format specified by the <code>fmt</code> date format.</td>
</tr>
<tr>
<td><code>TO_DSINTERVAL</code></td>
<td>Converts a character string of <code>CHAR</code>, <code>VARCHAR2</code>, <code>NCHAR</code>, or <code>NVARCHAR2</code> datatype to a value of <code>INTERVAL DAY TO SECOND</code> datatype.</td>
</tr>
<tr>
<td><code>TO_NCHAR</code> (datetime)</td>
<td>Converts a datetime or interval value of <code>DATE</code>, <code>TIMESTAMP</code>, <code>TIMESTAMP WITH TIME ZONE</code>, <code>TIMESTAMP WITH LOCAL TIME ZONE</code>, <code>INTERVAL MONTH TO YEAR</code>, or <code>INTERVAL DAY TO SECOND</code> datatype from the database character set to the national character set.</td>
</tr>
<tr>
<td><code>TO_TIMESTAMP</code></td>
<td>Converts a character string of <code>CHAR</code>, <code>VARCHAR2</code>, <code>NCHAR</code>, or <code>NVARCHAR2</code> datatype to a value of <code>TIMESTAMP</code> datatype.</td>
</tr>
<tr>
<td><code>TO_TIMESTAMP_TZ</code></td>
<td>Converts a character string of <code>CHAR</code>, <code>VARCHAR2</code>, <code>NCHAR</code>, or <code>NVARCHAR2</code> datatype to a value of the <code>TIMESTAMP WITH TIME ZONE</code> datatype.</td>
</tr>
<tr>
<td><code>TO_YMINTERVAL</code></td>
<td>Converts a character string of <code>CHAR</code>, <code>VARCHAR2</code>, <code>NCHAR</code>, or <code>NVARCHAR2</code> datatype to a value of the <code>INTERVAL YEAR TO MONTH</code> datatype.</td>
</tr>
<tr>
<td><code>TZ_OFFSET</code></td>
<td>Returns the time zone offset that corresponds to the entered value, based on the date that the statement is executed.</td>
</tr>
</tbody>
</table>

**See Also:** *Oracle Database SQL Reference*
Their default values are derived from NLS_TERRITORY.

You can specify their values by setting them in the initialization parameter file. You can specify their values for a client as client environment variables.

You can also change their values by changing their value in the initialization parameter file and then restarting the instance.

To change their values during a session, use the ALTER SESSION statement.

See Also:

- "Date and Time Parameters" on page 3-20 for more information, including examples
- "NLS_DATE_FORMAT" on page 3-21
- "NLS_TIMESTAMP_FORMAT" on page 3-24
- "NLS_TIMESTAMP_TZ_FORMAT" on page 3-25

### Time Zone Environment Variables

The time zone environment variables are:

- ORA_TZFILE, which specifies the Oracle time zone file used by the database
- ORA_SDTZ, which specifies the default session time zone

See Also:

- "Choosing a Time Zone File" on page 4-20
- "Setting the Session Time Zone" on page 4-24
Daylight Saving Time Session Parameter

**ERROR_ON_OVERLAP_TIME** is a session parameter that determines how Oracle handles an ambiguous datetime boundary value. Ambiguous datetime values can occur when the time changes between Daylight Saving Time and standard time.

The possible values are **TRUE** and **FALSE**. When **ERROR_ON_OVERLAP_TIME** is **TRUE**, then an error is returned when Oracle encounters an ambiguous datetime value. When **ERROR_ON_OVERLAP_TIME** is **FALSE**, then ambiguous datetime values are assumed to be standard time. The default value is **FALSE**.

*See Also: “Support for Daylight Saving Time” on page 4-26*

Choosing a Time Zone File

The Oracle time zone files contain the valid time zone names. The following information is also included for each time zone:

- Offset from Coordinated Universal Time (UTC)
- Transition times for Daylight Saving Time
- Abbreviations for standard time and Daylight Saving Time

Two time zone files are included in the Oracle home directory. The default time zone file, `$ORACLE_HOME/oracore/zoneinfo/timezone.dat`, contains the most commonly used time zones. More time zones are included in `$ORACLE_HOME/oracore/zoneinfo/timezlr.dat`.

To use the larger time zone file, complete the following tasks:

1. Shut down the database.
2. Set the `ORA_TZFILE` environment variable to the full path name of the `timezlr.dat` file.
3. Restart the database.

Oracle’s time zone data is derived from the public domain information available at `ftp://elsie.nci.nih.gov/pub/`. Oracle’s time zone data may not reflect the most recent data available at this site.

You can obtain a list of time zone names and time zone abbreviations from the time zone file that is installed with your database by entering the following statement:

```
SELECT tzname, tzabbrev FROM v$timezone_names;
```
For the default time zone file, this statement results in output similar to the following:

<table>
<thead>
<tr>
<th>TZNAME</th>
<th>TZABBREV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Africa/Cairo</td>
<td>LMT</td>
</tr>
<tr>
<td>Africa/Cairo</td>
<td>EET</td>
</tr>
<tr>
<td>Africa/Cairo</td>
<td>EEST</td>
</tr>
<tr>
<td>Africa/Tripoli</td>
<td>LMT</td>
</tr>
<tr>
<td>Africa/Tripoli</td>
<td>CET</td>
</tr>
<tr>
<td>Africa/Tripoli</td>
<td>CEST</td>
</tr>
<tr>
<td>Africa/Tripoli</td>
<td>EET</td>
</tr>
<tr>
<td>W-SU</td>
<td>LMT</td>
</tr>
<tr>
<td>W-SU</td>
<td>MMT</td>
</tr>
<tr>
<td>W-SU</td>
<td>MST</td>
</tr>
<tr>
<td>W-SU</td>
<td>MDST</td>
</tr>
<tr>
<td>W-SU</td>
<td>S</td>
</tr>
<tr>
<td>W-SU</td>
<td>MSD</td>
</tr>
<tr>
<td>W-SU</td>
<td>MSK</td>
</tr>
<tr>
<td>W-SU</td>
<td>EET</td>
</tr>
<tr>
<td>W-SU</td>
<td>EEST</td>
</tr>
<tr>
<td>WET</td>
<td>LMT</td>
</tr>
<tr>
<td>WET</td>
<td>WEST</td>
</tr>
<tr>
<td>WET</td>
<td>WET</td>
</tr>
</tbody>
</table>

622 rows selected.

There are 3 time zone abbreviations associated with the Africa/Cairo time zone and 4 abbreviations associated with the Africa/Tripoli time zone. The following table shows the time zone abbreviations and their meanings.

<table>
<thead>
<tr>
<th>Time Zone Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMT</td>
<td>Local Mean Time</td>
</tr>
<tr>
<td>EET</td>
<td>Eastern Europe Time</td>
</tr>
<tr>
<td>EEST</td>
<td>Eastern Europe Summer Time</td>
</tr>
<tr>
<td>CET</td>
<td>Central Europe Time</td>
</tr>
<tr>
<td>CEST</td>
<td>Central Europe Summer Time</td>
</tr>
</tbody>
</table>
Note that an abbreviation can be associated with more than one time zone. For example, EET is associated with both Africa/Cairo and Africa/Tripoli, as well as time zones in Europe.

If you want a list of time zones without repeating the time zone name for each abbreviation, use the following query:

```
SELECT UNIQUE tzname FROM v$timezone_names;
```

For the default file, this results in output similar to the following:

```
TZNAME
--------
Africa/Cairo
Africa/Tripoli
America/Adak
America/Anchorage
  .
  .
  .
US/Mountain
US/Pacific
US/Pacific_New
US/Samoa
W-SU
```

The default time zone file contains more than 180 unique time zone names. The large time zone file has more than 350 unique time zone names.

**Note:** If you use the larger time zone file, it is not practical to return to the smaller time zone file because the database may contain data with time zones that are not part of the smaller time zone file.

**See Also:**

- "Customizing Time Zone Data" on page 13-17
- "Time Zone Names" on page A-30 for a list of valid Oracle time zone names
Setting the Database Time Zone

Set the database time zone when the database is created by using the `SET TIME_ZONE` clause of the `CREATE DATABASE` statement. If you do not set the database time zone, it defaults to the time zone of the server’s operating system.

The time zone may be set to an absolute offset from UTC or to a named region. For example, to set the time zone to an offset from UTC, use a statement similar to the following:

```
CREATE DATABASE db01
...
SET TIME_ZONE='05:00';
```

The range of valid offsets is -12:00 to +14:00.

To set the time zone to a named region, use a statement similar to the following:

```
CREATE DATABASE db01
...
SET TIME_ZONE='Europe/London';
```

---

**Note:** The database time zone is relevant only for `TIMESTAMP WITH LOCAL TIME ZONE` columns. Oracle Corporation recommends that you set the database time zone to UTC (0:00) to avoid data conversion and improve performance when data is transferred among databases. This is especially important for distributed databases, replication, and exporting and importing.

---

You can change the database time zone by using the `SET TIME_ZONE` clause of the `ALTER DATABASE` statement. For example:

```
ALTER DATABASE SET TIME_ZONE='05:00';
ALTER DATABASE SET TIME_ZONE='Europe/London';
```

The `ALTER DATABASE SET TIME_ZONE` statement returns an error if the database contains a table with a `TIMESTAMP WITH LOCAL TIME ZONE` column and the column contains data.

The change does not take effect until the database has been shut down and restarted.

You can find out the database time zone by entering the following query:

```
SELECT dbtimezone FROM dual;
```
Setting the Session Time Zone

You can set the default session time zone with the ORA_SDTZ environment variable. When users retrieve TIMESTAMP WITH LOCAL TIME ZONE data, Oracle returns it in the users’ session time zone. The session time zone also takes effect when a TIMESTAMP value is converted to the TIMESTAMP WITH TIME ZONE or TIMESTAMP WITH LOCAL TIME ZONE datatype.

**Note:** Setting the session time zone does not affect the value returned by the SYSDATE and SYSTIMESTAMP SQL function. SYSDATE returns the date and time of the operating system on which the database resides, taking into account the time zone of the database server’s operating system that was in effect when the database was started.

The ORA_SDTZ environment variable can be set to the following values:

- Operating system local time zone (‘OS_TZ’)
- Database time zone (‘DB_TZ’)
- Absolute offset from UTC (for example, ‘-05:00’)
- Time zone region name (for example, ‘Europe/London’)

To set ORA_SDTZ, use statements similar to one of the following in a UNIX environment (C shell):

```
% setenv ORA_SDTZ ‘OS_TZ’
% setenv ORA_SDTZ ‘DB_TZ’
% setenv ORA_SDTZ ‘-05:00’
% setenv ORA_SDTZ ‘Europe/London’
```

You can change the time zone for a specific SQL session with the SET TIME_ZONE clause of the ALTER SESSION statement.

TIME_ZONE can be set to the following values:

- Default local time zone when the session was started (local)
- Database time zone (dftimezone)
- Absolute offset from UTC (for example, ‘+10:00’)
- Time zone region name (for example, ‘Asia/Hong_Kong’)

Use ALTER SESSION statements similar to the following:
Converting Time Zones With the AT TIME ZONE Clause

A datetime SQL expression can be one of the following:

- A datetime column
- A compound expression that yields a datetime value

A datetime expression can include an AT LOCAL clause or an AT TIME ZONE clause. If you include an AT LOCAL clause, then the result is returned in the current session time zone. If you include the AT TIME ZONE clause, then use one of the following settings with the clause:

- Time zone offset: The string ‘(+|-)HH:MM’ specifies a time zone as an offset from UTC. For example, ‘-07:00’ specifies the time zone that is 7 hours behind UTC. For example, if the UTC time is 11:00 a.m., then the time in the ‘-07:00’ time zone is 4:00 a.m.
- DBTIMEZONE: Oracle uses the database time zone established (explicitly or by default) during database creation.
- SESSIONTIMEZONE: Oracle uses the session time zone established by default or in the most recent ALTER SESSION statement.
- Time zone region name: Oracle returns the value in the time zone indicated by the time zone region name. For example, you can specify Asia/Hong_Kong.
- An expression: If an expression returns a character string with a valid time zone format, then Oracle returns the input in that time zone. Otherwise Oracle returns an error.

The following example converts the datetime value in the America/New_York time zone to the datetime value in the America/Los_Angeles time zone.

Example 4–5  Converting a Datetime Value to Another Time Zone

```
SELECT FROM_TZ(CAST(TO_DATE('1999-12-01 11:00:00', 'YYYY-MM-DD HH:MI:SS') AS TIMESTAMP), 'America/New_York')
```

Oracle automatically determines whether Daylight Saving Time is in effect for a specified time zone and returns the corresponding local time. The datetime value is usually sufficient for Oracle to determine whether Daylight Saving Time is in effect for a specified time zone. The periods when Daylight Saving Time begins or ends are boundary cases. For example, in the Eastern region of the United States, the time changes from 01:59:59 a.m. to 3:00:00 a.m. when Daylight Saving Time goes into effect. The interval between 02:00:00 and 02:59:59 a.m. does not exist. Values in that interval are invalid. When Daylight Saving Time ends, the time changes from 02:00:00 a.m. to 01:00:01 a.m. The interval between 01:00:01 and 02:00:00 a.m. is repeated. Values from that interval are ambiguous because they occur twice.

To resolve these boundary cases, Oracle uses the TZR and TZD format elements. TZR represents the time zone region in datetime input strings. Examples are 'Australia/North', 'UTC', and 'Singapore'. TZD represents an abbreviated form of the time zone region with Daylight Saving Time information. Examples are 'PST' for US/Pacific standard time and 'PDT' for US/Pacific daylight time. To see a list of valid values for the TZR and TZD format elements, query the TZNAME and TZABBREV columns of the V$TIMEZONE_NAMES dynamic performance view.

The rest of this section contains the following topic:

- Examples: The Effect of Daylight Saving Time on Datetime Calculations

See Also: "Time Zone Names" on page A-30 for a list of valid time zones

Examples: The Effect of Daylight Saving Time on Datetime Calculations

The TIMESTAMP datatype does not accept time zone values and does not calculate Daylight Saving Time.

The TIMESTAMP WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE datatypes have the following behavior:
If a time zone region is associated with the datetime value, then the database server knows the Daylight Saving Time rules for the region and uses the rules in calculations.

Daylight Saving Time is not calculated for regions that do not use Daylight Saving Time.

The rest of this section contains examples that use datetime datatypes. The examples use the `global_orders` table. It contains the `orderdate1` column of `TIMESTAMP` datatype and the `orderdate2` column of `TIMESTAMP WITH TIME ZONE` datatype. The `global_orders` table is created as follows:

```sql
CREATE TABLE global_orders ( orderdate1 TIMESTAMP(0),
    orderdate2 TIMESTAMP(0) WITH TIME ZONE);
```

```sql
INSERT INTO global_orders VALUES ( '28-OCT-00 11:24:54 PM',
    '28-OCT-00 11:24:54 PM America/New_York');
```

**Example 4-6 Comparing Daylight Saving Time Calculations Using TIMESTAMP WITH TIME ZONE and TIMESTAMP**

```sql
SELECT orderdate1 + INTERVAL '8' HOUR, orderdate2 + INTERVAL '8' HOUR
FROM global_orders;
```

The following output results:

<table>
<thead>
<tr>
<th>ORDERDATE1+INTERVAL’8’HOUR</th>
<th>ORDERDATE2+INTERVAL’8’HOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>29-OCT-00 07.24.54.000000 AM</td>
<td>29-OCT-00 06.24.54.000000 AM AMERICA/NEW_YORK</td>
</tr>
</tbody>
</table>

This example shows the effect of adding 8 hours to the columns. The time period includes a Daylight Saving Time boundary (a change from Daylight Saving Time to standard time). The `orderdate1` column is of `TIMESTAMP` datatype, which does not use Daylight Saving Time information and thus does not adjust for the change that took place in the 8-hour interval. The `TIMESTAMP WITH TIME ZONE` datatype does adjust for the change, so the `orderdate2` column shows the time as one hour earlier than the time shown in the `orderdate1` column.

**Note:** If you have created a `global_orders` table for the previous examples, then drop the `global_orders` table before you try Example 4–7 through Example 4–8.
Example 4–7   Comparing Daylight Saving Time Calculations Using TIMESTAMP WITH LOCAL TIME ZONE and TIMESTAMP

The TIMESTAMP WITH LOCAL TIME ZONE datatype uses the value of TIME_ZONE that is set for the session environment. The following statements set the value of the TIME_ZONE session parameter and create an orders table. The global_orders table has one column of TIMESTAMP datatype and one column of TIMESTAMP WITH LOCAL TIME ZONE datatype.

ALTER SESSION SET TIME_ZONE='America/New_York';
CREATE TABLE global_orders ( orderdate1 TIMESTAMP(0),
   orderdate2 TIMESTAMP(0) WITH LOCAL TIME ZONE );
INSERT INTO global_orders VALUES ( '28-OCT-00 11:24:54 PM',
   '28-OCT-00 11:24:54 PM' );

Add 8 hours to both columns.

SELECT orderdate1 + INTERVAL '8' HOUR, orderdate2 + INTERVAL '8' HOUR
   FROM global_orders;

Because a time zone region is associated with the datetime value for orderdate2, the Oracle server uses the Daylight Saving Time rules for the region. Thus the output is the same as in Example 4–6. There is a one-hour difference between the two calculations because Daylight Saving Time is not calculated for the TIMESTAMP datatype, and the calculation crosses a Daylight Saving Time boundary.

Example 4–8   Daylight Saving Time Is Not Calculated for Regions That Do Not Use Daylight Saving Time

Set the time zone region to UTC. UTC does not use Daylight Saving Time.

ALTER SESSION SET TIME_ZONE='UTC';

Truncate the global_orders table.
TRUNCATE TABLE global_orders;

Insert values into the global_orders table.

INSERT INTO global_orders VALUES ( '28-OCT-00 11:24:54 PM',
   TIMESTAMP '2000-10-28 23:24:54 ' );

Add 8 hours to the columns.

SELECT orderdate1 + INTERVAL '8' HOUR, orderdate2 + INTERVAL '8' HOUR
   FROM global_orders;

The following output results.
Support for Daylight Saving Time

ORDERDATE1+INTERVAL '8' HOUR

----------------------------
29-OCT-00 07.24.54.000000000 AM

ORDERDATE2+INTERVAL '8' HOUR

----------------------------
29-OCT-00 07.24.54.000000000 AM UTC

The times are the same because Daylight Saving Time is not calculated for the UTC time zone region.
This chapter explains sorting and searching for strings in an Oracle environment. It contains the following topics:

- Overview of Oracle’s Sorting Capabilities
- Using Binary Sorts
- Using Linguistic Sorts
- Linguistic Sort Features
- Using Linguistic Indexes
- Case-Insensitive and Accent-Insensitive Linguistic Sorts
- Searching Linguistic Strings
- SQL Regular Expressions in a Multilingual Environment
Overview of Oracle’s Sorting Capabilities

Different languages have different sort orders. In addition, different cultures or countries that use the same alphabets may sort words differently. For example, in Danish, Æ is after z, while ÿ and Ù are considered to be variants of the same letter.

Sort order can be case-sensitive or case-insensitive. Case refers to the condition of being uppercase or lowercase. For example, in a Latin alphabet, A is the uppercase glyph for a, the lowercase glyph.

Sort order can ignore or consider diacritics. A diacritic is a mark near or through a character or combination of characters that indicates a different sound than the sound of the character without the diacritic. For example, the cedilla (, ) in façade is a diacritic. It changes the sound of c.

Sort order can be phonetic or it can be based on the appearance of the character. For example, sort order can be based on the number of strokes in East Asian ideographs. Another common sorting issue is combining letters into a single character. For example, in traditional Spanish, ch is a distinct character that comes after c, which means that the correct order is: cerveza, colorado, cheremoya. This means that the letter c cannot be sorted until Oracle has checked whether the next letter is an h.

Oracle provides the following types of sorts:

- Binary sort
- Monolingual linguistic sort
- Multilingual linguistic sort

These sorts achieve a linguistically correct order for a single language as well as a sort based on the multilingual ISO standard (ISO 14651), which is designed to handle many languages at the same time.

Using Binary Sorts

One way to sort character data is based on the numeric values of the characters defined by the character encoding scheme. This is called a binary sort. Binary sorts are the fastest type of sort. They produce reasonable results for the English alphabet because the ASCII and EBCDIC standards define the letters A to Z in ascending numeric value.
When characters used in other languages are present, a binary sort usually does not produce reasonable results. For example, an ascending ORDER BY query returns the character strings \texttt{ABC}, \texttt{ABZ}, \texttt{BCD}, \texttt{ÄBC}, when \texttt{Ä} has a higher numeric value than \texttt{B} in the character encoding scheme. A binary sort is not usually linguistically meaningful for Asian languages that use ideographic characters.

**Using Linguistic Sorts**

To produce a sort sequence that matches the alphabetic sequence of characters, another sort technique must be used that sorts characters independently of their numeric values in the character encoding scheme. This technique is called a linguistic sort. A linguistic sort operates by replacing characters with numeric values that reflect each character’s proper linguistic order.

Oracle offers two kinds of linguistic sorts: monolingual and multilingual.

This section includes the following topics:

- Monolingual Linguistic Sorts
- Multilingual Linguistic Sorts
- Multilingual Sorting Levels
- Linguistic Sort Examples

**Monolingual Linguistic Sorts**

Oracle compares character strings in two steps for monolingual sorts. The first step compares the major value of the entire string from a table of major values. Usually, letters with the same appearance have the same major value. The second step compares the minor value from a table of minor values. The major and minor values are defined by Oracle. Oracle defines letters with diacritic and case differences as having the same major value but different minor values.

Each major table entry contains the **Unicode code point** and major value for a character. The Unicode code point is a 16-bit binary value that represents a character.

---

**Note:** In the ASCII standard, all uppercase letters appear before any lowercase letters. In the EBCDIC standard, the opposite is true: all lowercase letters appear before any uppercase letters.
Table 5–1 illustrates sample values for sorting a, A, ä, Ä, and b.

<table>
<thead>
<tr>
<th>Glyph</th>
<th>Major Value</th>
<th>Minor Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>A</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>ä</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Ä</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>b</td>
<td>20</td>
<td>5</td>
</tr>
</tbody>
</table>

See Also: “Overview of Unicode” on page 6-2

**Multilingual Linguistic Sorts**

Oracle provides multilingual linguistic sorts so that you can sort data in more than one language in one sort. This is useful for regions or languages that have complex sorting rules and for multilingual databases. Oracle Database 10g supports all of the sort orders defined by previous releases.

For Asian language data or multilingual data, Oracle provides a sorting mechanism based on the ISO 14651 standard and the Unicode 3.2 standard. Chinese characters are ordered by the number of strokes, PinYin, or radicals.

In addition, multilingual sorts can handle canonical equivalence and supplementary characters. **Canonical equivalence** is a basic equivalence between characters or sequences of characters. For example, ç is equivalent to the combination of c and ë. **Supplementary characters** are user-defined characters or predefined characters in Unicode 3.2 that require two code points within a specific code range. You can define up to 1.1 million code points in one multilingual sort.

For example, Oracle supports a monolingual French sort (FRENCH), but you can specify a multilingual French sort (FRENCH_M). _M represents the ISO 14651 standard for multilingual sorting. The sorting order is based on the GENERIC_M sorting order and can sort diacritical marks from right to left. Oracle Corporation recommends using a multilingual linguistic sort if the tables contain multilingual data. If the tables contain only French, then a monolingual French sort may have better performance because it uses less memory. It uses less memory because fewer characters are defined in a monolingual French sort than in a multilingual French sort. There is a tradeoff between the scope and the performance of a sort.
Using Linguistic Sorts

Linguistic Sorting and String Searching

5-5

Multilingual Sorting Levels

Oracle evaluates multilingual sorts at three levels of precision:

- Primary Level Sorts
- Secondary Level Sorts
- Tertiary Level Sorts

Primary Level Sorts

A primary level sort distinguishes between base letters, such as the difference between characters a and b. It is up to individual locales to define whether a is before b, b is before a, or if they are equal. The binary representation of the characters is completely irrelevant. If a character is an ignorable character, then it is assigned a primary level order (or weight) of zero, which means it is ignored at the primary level. Characters that are ignorable on other levels are given an order of zero at those levels.

For example, at the primary level, all variations of bat come before all variations of bet. The variations of bat can appear in any order, and the variations of bet can appear in any order:

Bat
bat
BAT
BET
Bet
bet

See Also: "Ignorable Characters" on page 5-7

Secondary Level Sorts

A secondary level sort distinguishes between base letters (the primary level sort) before distinguishing between diacritics on a given base letter. For example, the character Ä differs from the character A only because it has a diacritic. Thus, Ä and A are the same on the primary level because they have the same base letter (A) but differ on the secondary level.
The following list has been sorted on the primary level (resume comes before resumes) and on the secondary level (strings without diacritics come before strings with diacritics):

resume
résumé
Résumé
Resumes
resumes
résumés

**Tertiary Level Sorts**
A tertiary level sort distinguishes between base letters (primary level sort), diacritics (secondary level sort), and case (upper case and lower case). It can also include special characters such as +, -, and *.

The following are examples of tertiary level sorts:

- Characters a and A are equal on the primary and secondary levels but different on the tertiary level because they have different cases.
- Characters ä and A are equal on the primary level and different on the secondary and tertiary levels.
- The primary and secondary level orders for the dash character – is 0. That is, it is ignored on the primary and secondary levels. If a dash is compared with another character whose primary level order is nonzero, for example, u, then no result for the primary level is available because u is not compared with anything. In this case, Oracle finds a difference between – and u only at the tertiary level.

The following list has been sorted on the primary level (resume comes before resumes) and on the secondary level (strings without diacritics come before strings with diacritics) and on the tertiary level (lower case comes before upper case):

resume
Resume
résumé
Résumé
resumes
Resumes
résumés
Résumés
Linguistic Sort Features

This section contains information about different features that a linguistic sort can have:

- Base Letters
- Ignorable Characters
- Contracting Characters
- Expanding Characters
- Context-Sensitive Characters
- Canonical Equivalence
- Reverse Secondary Sorting
- Character Rearrangement for Thai and Laotian Characters
- Special Letters
- Special Combination Letters
- Special Uppercase Letters
- Special Lowercase Letters

You can customize linguistic sorts to include the desired characteristics.

See Also: Chapter 13, "Customizing Locale"

Base Letters

Base letters are defined in a base letter table, which maps each letter to its base letter. For example, a, A, ä, and Ä all map to a, which is the base letter. This concept is particularly relevant for working with Oracle Text.

See Also: Oracle Text Reference

Ignorable Characters

Some characters can be ignored in a linguistic sort. These characters are called ignorable characters. There are two kinds of ignorable characters: diacritics and punctuation.

Examples of ignorable diacritics are:

- ^, so that rôle is treated the same as role
The umlaut, so that naïve is treated the same as naive

An example of an ignorable punctuation character is the dash character -. If it is ignored, then multi-lingual can be treated that same as multilingual and e-mail can be treated the same as email.

**Contracting Characters**

Sorting elements usually consist of a single character, but in some locales, two or more characters in a character string must be considered as a single sorting element during sorting. For example, in traditional Spanish, the string ch is composed of two characters. These characters are called **contracting characters** in multilingual linguistic sorting and **special combination letters** in monolingual linguistic sorting.

Do not confuse a **composed character** with a contracting character. A composed character like á can be decomposed into a and ’, each with their own encoding. The difference between a composed character and a contracting character is that a composed character can be displayed as a single character on a terminal, while a contracting character is used only for sorting, and its component characters must be rendered separately.

**Expanding Characters**

In some locales, certain characters must be sorted as if they were character strings. An example is the German character ß (sharp s). It is sorted exactly the same as the string SS. Another example is that ö sorts as if it were oe, after od and before of. These characters are known as **expanding characters** in multilingual linguistic sorting and **special letters** in monolingual linguistic sorting. Just as with contracting characters, the replacement string for an expanding character is meaningful only for sorting.

**Context-Sensitive Characters**

In Japanese, a prolonged sound mark that resembles an em dash – represents a length mark that lengthens the vowel of the preceding character. The sort order depends on the vowel that precedes the length mark. This is called context-sensitive sorting. For example, after the character ka, the – length mark indicates a long a and is treated the same as a, while after the character ki, the – length mark indicates a long i and is treated the same as i. Transliterating this to Latin characters, a sort might look like this:

kaa
ka— -- kaa and ka— are the same
kai -- kai follows ka- because i is after a
kia -- kia follows kai because i is after a
ki-- -- kii and ki-- are the same

Canonical Equivalence

Canonical equivalence is an attribute of a multilingual sort and describes how equivalent code point sequences are sorted. If canonical equivalence is applied in a particular linguistic sort, then canonically equivalent strings are treated as equal.

One Unicode code point can be equivalent to a sequence of base letter code points plus diacritic code points. This is called the Unicode canonical equivalence. For example, ä equals its base letter a and an umlaut. A linguistic flag, CANONICAL_EQUIVALENCE=TRUE, indicates that all canonical equivalence rules defined in Unicode 3.2 need to be applied in a specific linguistic sort. Oracle-defined linguistic sorts include the appropriate setting for the canonical equivalence flag. You can set the flag to FALSE to speed up the comparison and ordering functions if all the data is in its composed form.

For example, consider the following strings:
- äa (a umlaut followed by a)
- a¨b (a followed by umlaut followed by b)
- äc (a umlaut followed by c)

If CANONICAL_EQUIVALENCE=FALSE, then the sort order of the strings is:
- a¨b
- äa
- äc

This occurs because a comes before ä if canonical equivalence is not applied.

If CANONICAL_EQUIVALENCE=TRUE, then the sort order of the strings is:
- äa
- a¨b
- äc

This occurs because ä and a¨ are treated as canonically equivalent.

You can use Oracle Locale Builder to view the setting of the canonical equivalence flag in existing multilingual sorts. When you create a customized multilingual sort with Oracle Locale Builder, you can set the canonical equivalence flag as desired.
Reverse Secondary Sorting

In French, sorting strings of characters with diacritics first compares base letters from left to right, but compares characters with diacritics from right to left. For example, by default, a character with a diacritic is placed after its unmarked variant. Thus Édit comes before Édit in a French sort. They are equal on the primary level, and the secondary order is determined by examining characters with diacritics from right to left. Individual locales can request that the characters with diacritics be sorted with the right-to-left rule. Set the REVERSE_SECONDARY linguistic flag to TRUE to enable reverse secondary sorting.

See Also: "Creating a New Linguistic Sort with the Oracle Locale Builder" on page 13-32 for more information about setting the reverse secondary flag

Character Rearrangement for Thai and Laotian Characters

In Thai and Lao, some characters must first change places with the following character before sorting. Normally, these types of character are symbols representing vowel sounds, and the next character is a consonant. Consonants and vowels must change places before sorting. Set the SWAP_WITH_NEXT linguistic flag for all characters that must change places before sorting.

See Also: "Creating a New Linguistic Sort with the Oracle Locale Builder" on page 13-32 for more information about setting the SWAP_WITH_NEXT flag

Special Letters

Special letters is a term used in monolingual sorts. They are called expanding characters in multilingual sorts.

See Also: "Expanding Characters" on page 5-8

Special Combination Letters

Special combination letters is the term used in monolingual sorts. They are called contracting letters in multilingual sorts.
Special Uppercase Letters

One lowercase letter may map to multiple uppercase letters. For example, in traditional German, the uppercase letters for ſ are SS.

These case conversions are handled by the NLS_UPPER, NLS_LOWER, and NLS_INITCAP SQL functions, according to the conventions established by the linguistic sort sequence. The UPPER, LOWER, and INITCAP SQL functions cannot handle these special characters.

The NLS_UPPER SQL function returns all uppercase characters from the same character set as the lowercase string. The following example shows the result of the NLS_UPPER function when NLS_SORT is set to XGERMAN:

```sql
SELECT NLS_UPPER ('große') "Uppercase" FROM DUAL;
```

Upper
-----
GROSSE

See Also: Oracle Database SQL Reference

Special Lowercase Letters

Oracle supports special lowercase letters. One uppercase letter may map to multiple lowercase letters. An example is the Turkish uppercase İ becoming a small, dotless i: ı.

Case-Insensitive and Accent-Insensitive Linguistic Sorts

Operation inside an Oracle database is always sensitive to the case and the accents (diacritics) of the characters. Sometimes you may need to perform case-insensitive or accent-insensitive comparisons and sorts.

In previous versions of the database, case-insensitive queries could be achieved by using the NLS_UPPER and NLS_LOWER SQL functions. The functions change the case of strings based on a specific linguistic sort definition. This enables you to perform case-insensitive searches regardless of the language being used. For example, create a table called test4 as follows:

```sql
SQL> CREATE TABLE test4(word VARCHAR2(12));
SQL> INSERT INTO test4 VALUES('GROSSE');
```

See Also: "Contracting Characters" on page 5-8
Case-Insensitive and Accent-Insensitive Linguistic Sorts

```sql
SQL> INSERT INTO test4 VALUES('Große');
SQL> INSERT INTO test4 VALUES('große');
SQL> SELECT * FROM test4;

WORD
------------
GROSSE
Große
große

Perform a case-sensitive search for GROSSE as follows:

SQL> SELECT word FROM test4 WHERE word='GROSSE';

WORD
------------
GROSSE

Perform a case-insensitive search for GROSSE using the NLS_UPPER function:

SELECT word FROM test4
WHERE NLS_UPPER(word, 'NLS_SORT = XGERMAN') = 'GROSSE';

WORD
------------
GROSSE
Große
große

Using NLS_UPPER and NLS_LOWER functions can be cumbersome because they need to be hard-coded into the application logic. A partial solution was introduced in Oracle9i Release 2 (9.2). It uses the GENERIC_BASELETTER linguistic sort. The GENERIC_BASELETTER sort groups all characters together based on their base letter values. This is achieved by ignoring their case and diacritic differences.

The following example shows a GENERIC_BASELETTER query. First create a table called test5:

CREATE TABLE test5(product VARCHAR2(20));
INSERT INTO test5 VALUES('DATABASE');
INSERT INTO test5 VALUES('dätäbase');
INSERT INTO test5 VALUES('database');
INSERT INTO test5 VALUES('Database');
SELECT product FROM test5;

PRODUCT
```
Database

dätäbase
database
Database

Select **database** from **test5**, using the default binary sort:

```
SELECT product FROM test5 WHERE product='database';
```

**PRODUCT**

-----------------------
database

Set **NLS_COMP** to **ANSI** to perform a linguistic sort based on the value of **NLS_SORT**:

```
ALTER SESSION SET NLS_COMP=ANSI;
```

Set **NLS_SORT** to **GENERIC_BASELETTER**:

```
ALTER SESSION SET NLS_SORT=GENERIC_BASELETTER;
```

Again select **database** from **test5**:

```
SELECT * FROM test5 WHERE product='database';
```

**PRODUCT**

-----------------------
DATABASE
dätäbase
database
Database

Note that all of the rows of **test5** are selected.

The **GENERIC_BASELETTER** sort defines the base letters of the underlying characters. Hence it simulates the behavior of a case-insensitive and accent-insensitive linguistic sort. However, the **GENERIC_BASELETTER** search is not a linguistically sensitive search because it is not based on any specific language.

In Oracle Database 10g, Oracle provides case-insensitive and accent-insensitive options for linguistic sorts.

Oracle provides the following types of monolingual and multilingual linguistic sorts:
Linguistic sorts that use information about base letters, diacritics, punctuation, and case. These are the standard monolingual and multilingual linguistic sorts that are described in "Using Linguistic Sorts" on page 5-3.

Linguistic sorts that use information about base letters, diacritics, and punctuation. This type of sort is called **case-insensitive**.

Linguistic sorts that use information about base letters only. This type of sort is called **accent-insensitive**. (**Accent** is another word for **diacritic**.) An accent-insensitive sort is always case-insensitive as well.

The rest of this section contains the following topics:

- Examples of Case-Insensitive and Accent-Insensitive Sorts
- Specifying a Case-Insensitive or Accent-Insensitive Sort

**See Also:**
- "NLS_SORT" on page 3-41
- "NLS_COMP" on page 3-42

### Examples of Case-Insensitive and Accent-Insensitive Sorts

The following examples show:

- A sort that uses information about base letters, diacritics, punctuation, and case
- A case-insensitive sort
- An accent-insensitive sort

**Example 5–1 Linguistic Sort Using Base Letters, Diacritics, Punctuation, and Case Information**

The following list has been sorted using information about base letters, diacritics, punctuation, and case:

blackbird
Blackbird
black bird
black-bird
Black-bird
bläckbird
blackbird
**Example 5–2  Case-Insensitive Linguistic Sort**

The following list has been sorted using information about base letters, diacritics, and punctuation, ignoring case:

Blackbird
blackbird
bläckbird
blackbird
black bird
black-bird
Black-bird
bläckbird and Black-bird have the same value in the sort, because the only different between them is case. They could appear interchanged in the list.

Blackbird and blackbird also have the same value in the sort and could appear interchanged in the list.

**Example 5–3  Accent-Insensitive Linguistic Sort**

The following list has been sorted using information about base letters only. No information about diacritics, punctuation, or case has been used.

blackbird
bläckbird
blackbird
Blackbird
BlackBird
Black-bird
Black bird

**Specifying a Case-Insensitive or Accent-Insensitive Sort**

Use the `NLS_SORT` session parameter to specify a case-insensitive or accent-insensitive sort:

- Append `_CI` to an Oracle sort name for a case-insensitive sort.
- Append `_AI` to an Oracle sort name for an accent-insensitive and case-insensitive sort.

For example, you can set `NLS_SORT` to the following types of values:

FRENCH_M_AI
XGERMAN_CI
Binary sorts can also be case-insensitive or accent-insensitive. When you specify
BINARY_CI as a value for NLS_SORT, it designates a sort that is accent-sensitive
and case-insensitive. BINARY_AI designates an accent-insensitive and
case-insensitive binary sort. You may want to use a binary sort if the binary sort
order of the character set is appropriate for the character set you are using.

For example, with the NLS_LANG environment variable set to AMERICAN_ AMERICA.WE8ISO8859P1, create a table called test1 and populate it as follows:

```
SQL> CREATE TABLE test1 (letter VARCHAR2(10));
SQL> INSERT INTO test1 VALUES('ä');
SQL> INSERT INTO test1 VALUES('a');
SQL> INSERT INTO test1 VALUES('A');
SQL> INSERT INTO test1 VALUES('Z');
SQL> SELECT * FROM test1;
```

```
LETTER
------
ä
a
A
Z
```

The default value of NLS_SORT is BINARY. Use the following statement to do a
binary sort of the characters in table test1:

```
SELECT * FROM test1 ORDER BY letter;
```

To change the value of NLS_SORT, enter a statement similar to the following:

```
ALTER SESSION SET NLS_SORT=BINARY_CI;
```

The following table shows the sort orders that result from setting NLS_SORT to
BINARY, BINARY_CI, and BINARY_AI.

<table>
<thead>
<tr>
<th>BINARY</th>
<th>BINARY_CI</th>
<th>BINARY_AI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>a</td>
<td>ä</td>
</tr>
<tr>
<td>Z</td>
<td>A</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>Z</td>
<td>A</td>
</tr>
<tr>
<td>ä</td>
<td>ä</td>
<td>Z</td>
</tr>
</tbody>
</table>
When `NLS_SORT=BINARY`, uppercase letters come before lowercase letters. Letters with diacritics appear last.

When the sort considers diacritics but ignores case (`BINARY_CI`), the letters with diacritics appear last.

When both case and diacritics are ignored (`BINARY_AI`), ä is sorted with the other characters whose base letter is a. All the characters whose base letter is a occur before z.

You can use binary sorts for better performance when the character set is US7ASCII or another character set that has the same sort order as the binary sorts.

The following table shows the sort orders that result from German sorts for the table.

<table>
<thead>
<tr>
<th>GERMAN</th>
<th>GERMAN_CI</th>
<th>GERMAN_AI</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>ä</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>a</td>
</tr>
<tr>
<td>ä</td>
<td>ä</td>
<td>A</td>
</tr>
<tr>
<td>z</td>
<td>z</td>
<td>z</td>
</tr>
</tbody>
</table>

A German sort places lowercase letters before uppercase letters, and ä occurs before Z. When the sort ignores both case and diacritics (`GERMAN_AI`), ä appears with the other characters whose base letter is a.

**Linguistic Sort Examples**

The examples in this section demonstrate a binary sort, a monolingual sort, and a multilingual sort. To prepare for the examples, create and populate a table called `test2`. Enter the following statements:

```
SQL> CREATE TABLE test2 (name VARCHAR2(20));
SQL> INSERT INTO test2 VALUES('Diet');
SQL> INSERT INTO test2 VALUES('À voir');
SQL> INSERT INTO test2 VALUES('Freizeit');
```

**Example 5–4 Binary Sort**

The `ORDER BY` clause uses a binary sort.

```
SQL> SELECT * FROM test2 ORDER BY name;
```
You should see the following output:

Diet
Freizeit
À voir

Note that a binary sort results in À voir being at the end of the list.

**Example 5–5  Monolingual German Sort**

Use the NLSSORT function with the NLS_SORT parameter set to german to obtain a German sort.

```
SQL> SELECT * FROM test2 ORDER BY NLSSORT(name, 'NLS_SORT=german');
```

You should see the following output:

À voir
Diet
Freizeit

Note that À voir is at the beginning of the list in a German sort.

**Example 5–6  Comparing a Monolingual German Sort to a Multilingual Sort**

Insert the character string shown in Figure 5–1 into test. It is a D with a crossbar followed by ñ.

**Figure 5–1  Character String**

Déñ

Perform a monolingual German sort by using the NLSSORT function with the NLS_SORT parameter set to german.

```
SQL> SELECT * FROM test2 ORDER BY NLSSORT(name, 'NLS_SORT=german');
```

The output from the German sort shows the new character string last in the list of entries because the characters are not recognized in a German sort.

Perform a multilingual sort by entering the following statement:

```
SQL> SELECT * FROM test2 ORDER BY NLSSORT(name, 'NLS_SORT=generic_m');
```

The output shows the new character string after Diet, following ISO sorting rules.
Using Linguistic Indexes

Linguistic sorting is language-specific and requires more data processing than binary sorting. Using a binary sort for ASCII is accurate and fast because the binary codes for ASCII characters reflect their linguistic order. When data in multiple languages is stored in the database, you may want applications to sort the data returned from a `SELECT...ORDER BY` statement according to different sort sequences depending on the language. You can accomplish this without sacrificing performance by using linguistic indexes. Although a linguistic index for a column slows down inserts and updates, it greatly improves the performance of linguistic sorting with the `ORDER BY` clause.

You can create a function-based index that uses languages other than English. The index does not change the linguistic sort order determined by `NLS_SORT`. The index simply improves the performance. The following statement creates an index based on a German sort:

```sql
CREATE TABLE my_table(name VARCHAR(20) NOT NULL) /*NOT NULL ensures that the index is used */
CREATE INDEX nls_index ON my_table (NLSSORT(name, 'NLS_SORT = German'));
```

After the index has been created, enter a `SELECT` statement similar to the following:

```sql
SELECT * FROM my_table ORDER BY name;
```

It returns the result much faster than the same `SELECT` statement without an index.

The rest of this section contains the following topics:

- Linguistic Indexes for Multiple Languages
- Requirements for Using Linguistic Indexes

See Also:

- "The NLSSORT Function" on page 9-9
- "NLS_SORT" on page 3-41 for more information about setting and changing the NLS_SORT parameter

See Also:

- Oracle Database Concepts
- Oracle Database SQL Reference for more information about function-based indexes
Linguistic Indexes for Multiple Languages

There are three ways to build linguistic indexes for data in multiple languages:

- Build a linguistic index for each language that the application supports. This approach offers simplicity but requires more disk space. For each index, the rows in the language other than the one on which the index is built are collated together at the end of the sequence. The following example builds linguistic indexes for French and German.

```
CREATE INDEX french_index ON employees (NLSSORT(employee_id, 'NLS_SORT=FRENCH'));
CREATE INDEX german_index ON employees (NLSSORT(employee_id, 'NLS_SORT=GERMAN'));
```

Oracle chooses the index based on the NLS_SORT session parameter or the arguments of the NLSSORT function specified in the ORDER BY clause. For example, if the NLS_SORT session parameter is set to FRENCH, then Oracle uses french_index. When it is set to GERMAN, Oracle uses german_index.

- Build a single linguistic index for all languages. This requires a language column (LANG_COL in "Example: Setting Up a French Linguistic Index" on page 5-21) to be used as a parameter of the NLSSORT function. The language column contains NLS_LANGUAGE values for the data in the column on which the index is built. The following example builds a single linguistic index for multiple languages. With this index, the rows with the same values for NLS_LANGUAGE are sorted together.

```
CREATE INDEX i ON t (NLSSORT(col, 'NLS_SORT=' || LANG_COL));
```

Queries choose an index based on the argument of the NLSSORT function specified in the ORDER BY clause.

- Build a single linguistic index for all languages using one of the multilingual linguistic sorts such as GENERIC_M or FRENCH_M. These indexes sort characters according to the rules defined in ISO 14651. For example:

```
CREATE INDEX i on t (NLSSORT(col, 'NLS_SORT=GENERIC_M'));
```

See Also: "Multilingual Linguistic Sorts" on page 5-4 for more information about Unicode sorts

Requirements for Using Linguistic Indexes

The following are requirements for using linguistic indexes:
Set NLS_SORT Appropriately

The NLS_SORT parameter should indicate the linguistic definition you want to use for the linguistic sort. If you want a French linguistic sort order, then NLS_SORT should be set to FRENCH. If you want a German linguistic sort order, then NLS_SORT should be set to GERMAN.

There are several ways to set NLS_SORT. You should set NLS_SORT as a client environment variable so that you can use the same SQL statements for all languages. Different linguistic indexes can be used when NLS_SORT is set in the client environment.

See Also:  "NLS_SORT" on page 3-41

Specify NOT NULL in a WHERE Clause If the Column Was Not Declared NOT NULL

When you want to use the ORDER BY column_name clause with a column that has a linguistic index, include a WHERE clause like the following:

WHERE NLSSORT(column_name) IS NOT NULL

This WHERE clause is not necessary if the column has already been defined as a NOT NULL column in the schema.

Example: Setting Up a French Linguistic Index

The following example shows how to set up a French linguistic index. You may want to set NLS_SORT as a client environment variable instead of using the ALTER SESSION statement.

```
ALTER SESSION SET NLS_SORT='FRENCH';
ALTER SESSION SET OPTIMIZER_MODE = FIRST_ROWS;
CREATE INDEX test_idx ON test3(NLSSORT(col, 'NLS_SORT=FRENCH'));
SELECT * FROM test3 ORDER BY col;
ALTER SESSION SET NLS_COMP=ANSI;
SELECT * FROM test3 WHERE col > 'JJJ';
```
Searching Linguistic Strings

Searching and sorting are related tasks. Organizing data and processing it in a linguistically meaningful order is necessary for proper business processing. Searching and matching data in a linguistically meaningful way depends on what sort order is applied. For example, searching for all strings greater than \texttt{c} and less than \texttt{f} produces different results depending on the value of \texttt{NLS_SORT}. In a ASCII binary sort the search finds any strings that start with \texttt{d} or \texttt{e} but excludes entries that begin with upper case \texttt{D} or \texttt{E} or accented \texttt{e} with a diacritic, such as \texttt{ê}. Applying an accent-insensitive binary sort returns all strings that start with \texttt{d}, \texttt{D}, and accented \texttt{e}, such as \texttt{Ê} or \texttt{ê}. Applying the same search with \texttt{NLS_SORT} set to \texttt{XSPANISH} also returns strings that start with \texttt{ch}, because \texttt{ch} is treated as a composite character that sorts between \texttt{c} and \texttt{d} in traditional Spanish. This chapter discusses the kinds of sorts that Oracle offers and how they affect string searches by SQL and SQL regular expressions.

See Also:
- "Linguistic Sort Features" on page 5-7
- "SQL Regular Expressions in a Multilingual Environment" on page 5-22

SQL Regular Expressions in a Multilingual Environment

Regular expressions provide a powerful method of identifying patterns of strings within a body of text. Usage ranges from a simple search for a string such as \texttt{San Francisco} to the more complex task of extracting all URLs to a task like finding all words whose every second character is a vowel. SQL and PL/SQL support regular expressions in Oracle Database 10g.

Traditional regular expression engines were designed to address only English text. However, regular expression implementations can encompass a wide variety of languages with characteristics that are very different from western European text. Oracle’s implementation of regular expressions is based on the Unicode Regular Expression Guidelines. The \texttt{REGEXP} SQL functions work with all character sets that are supported as database character sets and national character sets. Moreover, Oracle enhances the matching capabilities of the POSIX regular expression constructs to handle the unique linguistic requirements of matching multilingual data.

Oracle enhancements of the linguistic-sensitive operators are described in the following sections:
Character Range \([x-y]\) in Regular Expressions

According to the POSIX standard, a range in a regular expression includes all collation elements between the start point and the end point of the range in the linguistic definition of the current locale. Therefore, ranges in regular expressions are meant to be linguistic ranges, not byte value ranges, because byte value ranges depend on the platform, and the end user should not be expected to know the ordering of the byte values of the characters. The semantics of the range expression must be independent of the character set. This implies that a range such as \([a-d]\) includes all the letters between \(a\) and \(d\) plus all of those letters with diacritics, plus any special case collation element such as \(\text{ch}\) in Traditional Spanish that is sorted as one character.

Oracle interprets range expressions as specified by the NLS_SORT parameter to determine the collation elements covered by a given range. For example:

```
Expression:     [a-d]e
NLS_SORT:       BINARY
Does not match: cheremoya
NLS_SORT:       XSPANISH
Matches:        >>che<<remoya
```

Collation Element Delimiter \([. .]\) in Regular Expressions

This construct is introduced by the POSIX standard to separate collating elements. A collating element is a unit of collation and is equal to one character in most cases. However, the collation sequence in some languages may define two or more characters as a collating element. The historical regular expression syntax does not
allow the user to define ranges involving multicharacter collation elements. For example, there was no way to define a range from a to ch because ch was interpreted as two separate characters.

By using the collating element delimiter [. .], you can separate a multicharacter collation element from other elements. For example, the range from a to ch can be written as [a- [. ch .]]. It can also be used to separate single-character collating elements. If you use [. .] to enclose a multicharacter sequence that is not a defined collating element, then it is considered as a semantic error in the regular expression. For example, [. ab .] is considered invalid if ab is not a defined multicharacter collating element.

**Character Class ’[: :]’ in Regular Expressions**

In English regular expressions, the range expression can be used to indicate a character class. For example, [a-z] can be used to indicate any lowercase letter. However, in non-English regular expressions, this approach is not accurate unless a is the first lowercase letter and z is the last lowercase letter in the collation sequence of the language.

The POSIX standard introduces a new syntactical element to enable specifying explicit character classes in a portable way. The [: :] syntax denotes the set of characters belonging to a certain character class. The character class definition is based on the character set classification data.

**Equivalence Class ’[= =]’ in Regular Expressions**

Oracle also supports equivalence classes through the [= =] syntax as recommended by the POSIX standard. A base letter and all of the accented versions of the base constitute an equivalence class. For example, the equivalence class [=a=] matches ä as well as å. The current implementation does not support matching of Unicode composed and decomposed forms for performance reasons. For example, ä (a umlaut) does not match ‘a followed by umlaut’.

**Examples: Regular Expressions**

The following examples show regular expression matches.

**Example 5–7 Case-Insensitive Match Using the NLS_SORT Value**

Case sensitivity in an Oracle regular expression match is determined at two levels: the NLS_SORT initialization parameter and the runtime match option. The REGEXP functions inherit the case-sensitivity behavior from the value of NLS_SORT by
The value can also be explicitly overridden by the runtime match option
‘c’ (case sensitive) or ‘i’ (case insensitive).

Expression: catalog(ue)?
NLS_SORT: GENERIC_M_CI
Matches:
   >>Catalog<<
   >>catalogue<<
   >>CATALOG<<

Oracle SQL syntax:
SQL> ALTER SESSION SET NLS_SORT='GENERIC_M_CI';
SQL> SELECT col FROM test WHERE REGEXP_LIKE(col,'catalog(ue)?');

**Example 5–8  Case Insensitivity Overridden by the Runtime Match Option**

Expression: catalog(ue)?
NLS_SORT: GENERIC_M_CI
Match option: ‘c’
Matches:
   >>catalogue<<
Does not match:
   Catalog
   CATALOG

Oracle SQL syntax:
SQL> ALTER SESSION SET NLS_SORT='GENERIC_M_CI';
SQL> SELECT col FROM test WHERE REGEXP_LIKE(col,'catalog(ue)?','c');

**Example 5–9  Matching with the Collation Element Operator [.]**

Expression: [^a-[:ch.]]+
Matches:
   >>driver<<
Does not match:
   cab

Oracle SQL syntax:
SQL> SELECT col FROM test WHERE REGEXP_LIKE(col,'[^a-[:ch.]]+');

**Example 5–10  Matching with the Character Class Operator [:]**

This expression looks for 6-character strings with lowercase characters. Note that
accented characters are matched as lowercase characters.
Expression: 

```
[:lower:]{6}
```

Database character set: WE8ISO1559P1

Matches:
```
>>maître<<
>>mòbile<<
>>pájaro<<
>>zurück<<
```

Oracle SQL syntax:

```
SQL> SELECT col FROM test WHERE REGEXP_LIKE(col,'[:lower:]{6}');
```

**Example 5–11   Matching with the Base Letter Operator [==]**

Expression: r[[=e=]sum[[=e=]]

Matches:
```
>>resume<<
>>résumé<<
>>résume<<
>>resumé<<
```

Oracle SQL syntax:

```
SQL> SELECT col FROM test WHERE REGEXP_LIKE(col,r[[=e=]sum[[e=]]');
```

**See Also:**

- *Oracle Database Application Developer’s Guide - Fundamentals* for more information about regular expression syntax
- *Oracle Database SQL Reference* for more information about REGEX SQL functions
Supporting Multilingual Databases with Unicode

This chapter illustrates how to use Unicode in an Oracle database environment. It includes the following topics:

- Overview of Unicode
- What is Unicode?
- Implementing a Unicode Solution in the Database
- Unicode Case Studies
- Designing Database Schemas to Support Multiple Languages
Overview of Unicode

Dealing with many different languages in the same application or database has been complicated and difficult for a long time. To overcome the limitations of existing character encodings, several organizations began working on the creation of a global character set in the late 1980s. The need for this became even greater with the development of the World Wide Web in the mid-1990s. The Internet has changed how companies do business, with an emphasis on the global market that has made a universal character set a major requirement. A global character set needs to fulfill the following conditions:

- Contain all major living scripts
- Support legacy data and implementations
- Be simple enough that a single implementation of an application is sufficient for worldwide use

A global character set should also have the following capabilities:

- Support multilingual users and organizations
- Conform to international standards
- Enable worldwide interchange of data

This global character set exists, is in wide use, and is called Unicode.

What is Unicode?

Unicode is a universal encoded character set that enables information from any language to be stored using a single character set. Unicode provides a unique code value for every character, regardless of the platform, program, or language.

The Unicode standard has been adopted by many software and hardware vendors. Many operating systems and browsers now support Unicode. Unicode is required by standards such as XML, Java, JavaScript, LDAP, and WML. It is also synchronized with the ISO/IEC 10646 standard.

Oracle Corporation started supporting Unicode as a database character set in version 7. In Oracle Database 10g, Unicode support has been expanded. Oracle Database 10g supports Unicode 3.2.

See Also: http://www.unicode.org for more information about the Unicode standard
What is Unicode?

This section contains the following topics:

- Supplementary Characters
- Unicode Encodings
- Oracle’s Support for Unicode

Supplementary Characters

The first version of Unicode was a 16-bit, fixed-width encoding that used two bytes to encode each character. This allowed 65,536 characters to be represented. However, more characters need to be supported, especially additional CJK ideographs that are important for the Chinese, Japanese, and Korean markets.

Unicode 3.2 defines supplementary characters to meet this need. It uses two 16-bit code points (also known as supplementary characters) to represent a single character. This enables an additional 1,048,576 characters to be defined. The Unicode 3.2 standard defines 45,960 supplementary characters.

Adding supplementary characters increases the complexity of Unicode, but it is less complex than managing several different encodings in the same configuration.

Unicode Encodings

Unicode 3.2 encodes characters in different ways: UTF-8, UCS-2, and UTF-16. Conversion between different Unicode encodings is a simple bit-wise operation that is defined in the Unicode standard.

This section contains the following topics:

- UTF-8 Encoding
- UCS-2 Encoding
- UTF-16 Encoding
- Examples: UTF-16, UTF-8, and UCS-2 Encoding

UTF-8 Encoding

UTF-8 is the 8-bit encoding of Unicode. It is a variable-width encoding and a strict superset of ASCII. This means that each and every character in the ASCII character set is available in UTF-8 with the same code point values. One Unicode character can be 1 byte, 2 bytes, 3 bytes, or 4 bytes in UTF-8 encoding. Characters from the European scripts are represented in either 1 or 2 bytes. Characters from most Asian...
scripts are represented in 3 bytes. Supplementary characters are represented in 4 bytes.

UTF-8 is the Unicode encoding supported on UNIX platforms and used for HTML and most Internet browsers. Other environments such as Windows and Java use UCS-2 encoding.

The benefits of UTF-8 are as follows:

- Compact storage requirement for European scripts because it is a strict superset of ASCII
- Ease of migration between ASCII-based characters sets and UTF-8

See Also:
- "Supplementary Characters" on page 6-3
- Table B–2, "Unicode Character Code Ranges for UTF-8 Character Codes" on page B-2

UCS-2 Encoding

UCS-2 is a fixed-width, 16-bit encoding. Each character is 2 bytes. UCS-2 is the Unicode encoding used by Java and Microsoft Windows NT 4.0. UCS-2 supports characters defined for Unicode 3.0, so there is no support for supplementary characters.

The benefits of UCS-2 over UTF-8 are as follows:

- More compact storage for Asian scripts because all characters are two bytes
- Faster string processing because characters are fixed-width
- Better compatibility with Java and Microsoft clients

See Also: "Supplementary Characters" on page 6-3

UTF-16 Encoding

UTF-16 encoding is the 16-bit encoding of Unicode. UTF-16 is an extension of UCS-2 because it supports the supplementary characters that are defined in Unicode 3.2 by using two UCS-2 code points for each supplementary character. UTF-16 is a strict superset of UCS-2.

One character can be either 2 bytes or 4 bytes in UTF-16. Characters from European and most Asian scripts are represented in 2 bytes. Supplementary characters are
What is Unicode?

represented in 4 bytes. UTF-16 is the main Unicode encoding used by Microsoft Windows 2000.

The benefits of UTF-16 over UTF-8 are as follows:

- More compact storage for Asian scripts because most of the commonly used Asian characters are represented in two bytes.
- Better compatibility with Java and Microsoft clients

See Also:

- "Supplementary Characters" on page 6-3
- Table B–1, "Unicode Character Code Ranges for UTF-16 Character Codes" on page B-2

Examples: UTF-16, UTF-8, and UCS-2 Encoding

Figure 6–1 shows some characters and their character codes in UTF-16, UTF-8, and UCS-2 encoding. The last character is a treble clef (a music symbol), a supplementary character that has been added to the Unicode 3.2 standard.

**Figure 6–1 UTF-16, UTF-8, and UCS-2 Encoding Examples**

<table>
<thead>
<tr>
<th>Character</th>
<th>UTF-16</th>
<th>UTF-8</th>
<th>UCS-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0041</td>
<td>41</td>
<td>0041</td>
</tr>
<tr>
<td>c</td>
<td>0063</td>
<td>63</td>
<td>0063</td>
</tr>
<tr>
<td>Ö</td>
<td>00F6</td>
<td>C3 B6</td>
<td>00F6</td>
</tr>
<tr>
<td>亞</td>
<td>4E9C</td>
<td>E4 BA 9C</td>
<td>4E9C</td>
</tr>
<tr>
<td>☇</td>
<td>D834 DD1E</td>
<td>F0 9D 84 9E</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Oracle’s Support for Unicode

Oracle Corporation started supporting Unicode as a database character set in version 7. Table 6–1 summarizes the Unicode character sets supported by the Oracle database server.
Implementing a Unicode Solution in the Database

You can store Unicode characters in an Oracle database in two ways.

You can create a Unicode database that enables you to store UTF-8 encoded characters as SQL CHAR datatypes (CHAR, VARCHAR2, CLOB, and LONG).

---

Table 6-1 Unicode Character Sets Supported by the Oracle Database Server

<table>
<thead>
<tr>
<th>Character Set</th>
<th>Supported in RDBMS Release</th>
<th>Unicode Encoding</th>
<th>Unicode Version</th>
<th>Database Character Set</th>
<th>National Character Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL24UTFFSS</td>
<td>7.2 - 8i</td>
<td>UTF-8</td>
<td>1.1</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>UTF8</td>
<td>8.0 - 10g</td>
<td>UTF-8</td>
<td>For Oracle release 8.0 through Oracle8i release 8.1.6: 2.1</td>
<td>Yes</td>
<td>Yes (Oracle9i and Oracle Database 10g only)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UTF-EBCDIC</td>
<td>For Oracle8i releases 8.0 through 8.1.6: 2.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For Oracle8i release 8.1.7 and later: 3.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UTFE</td>
<td>8.0 - 10g</td>
<td>UTF-EBCDIC</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>AL32UTF8</td>
<td>9i - 10g</td>
<td>UTF-8</td>
<td>Oracle9i, Release 1: 3.0</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Oracle9i, Release 2: 3.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Oracle Database 10g, Release 1: 3.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AL16UTF16</td>
<td>9i - 10g</td>
<td>UTF-16</td>
<td>Oracle9i, Release 1: 3.0</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Oracle9i, Release 2: 3.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Oracle Database 10g, Release 1: 3.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementing a Unicode Solution in the Database

If you prefer to implement Unicode support incrementally or if you need to support multilingual data only in certain columns, then you can store Unicode data in either the UTF-16 or UTF-8 encoding form in SQL NCHAR datatypes (NCHAR, NVARCHAR2, and NCLOB). The SQL NCHAR datatypes are called Unicode datatypes because they are used only for storing Unicode data.

**Note:** You can combine a Unicode database solution with a Unicode datatype solution.

The following sections explain how to use the two Unicode solutions and how to choose between them:

- Enabling Multilingual Support with Unicode Databases
- Enabling Multilingual Support with Unicode Datatypes
- How to Choose Between a Unicode Database and a Unicode Datatype Solution
- Comparing Unicode Character Sets for Database and Datatype Solutions

## Enabling Multilingual Support with Unicode Databases

The database character set specifies the encoding to be used in the SQL CHAR datatypes as well as the metadata such as table names, column names, and SQL statements. A **Unicode database** is a database with a UTF-8 character set as the database character set. There are three Oracle character sets that implement the UTF-8 encoding. The first two are designed for ASCII-based platforms while the third one should be used on EBCDIC platforms.

- **AL32UTF8**
  
  The AL32UTF8 character set supports the latest version of the Unicode standard. It encodes characters in one, two, or three bytes. Supplementary characters require four bytes. It is for ASCII-based platforms.

- **UTF8**
  
  The UTF8 character set encodes characters in one, two, or three bytes. It is for ASCII-based platforms.

  The UTF8 character set has supported Unicode 3.0 since Oracle8i release 8.1.7 and will continue to support Unicode 3.0 in future releases of the Oracle database server. Although specific supplementary characters were not assigned code points in Unicode until version 3.1, the code point range was allocated for...
supplementary characters in Unicode 3.0. If supplementary characters are inserted into a UTF8 database, then it does not corrupt the data in the database. The supplementary characters are treated as two separate, user-defined characters that occupy 6 bytes. Oracle Corporation recommends that you switch to AL32UTF8 for full support of supplementary characters in the database character set.

- UTFE

The UTFE character set is for EBCDIC platforms. It is similar to UTF8 on ASCII platforms, but it encodes characters in one, two, three, and four bytes. Supplementary characters are converted as two 4-byte characters.

**Example 6–1 Creating a Database with a Unicode Character Set**

To create a database with the AL32UTF8 character set, use the CREATE DATABASE statement and include the CHARACTER SET AL32UTF8 clause. For example:

```sql
CREATE DATABASE sample
  CONTROLFILE REUSE
  LOGFILE
    GROUP 1 ('diskx:log1.log', 'disky:log1.log') SIZE 50K,
    GROUP 2 ('diskx:log2.log', 'disky:log2.log') SIZE 50K
MAXLOGFILES 5
MAXLOGHISTORY 100
MAXDATAFILES 10
MAXINSTANCES 2
ARCHIVELOG
CHARACTER SET AL32UTF8
NATIONAL CHARACTER SET AL16UTF16
DATAFILE
  'disk1:df1.dbf' AUTOEXTEND ON,
  'disk2:df2.dbf' AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED
DEFAULT TEMPORARY TABLESPACE temp_ts
UNDO TABLESPACE undo_ts
SET TIME_ZONE = ' +02:00' ;
```

**Note:** Specify the database character set when you create the database.

---

**Enabling Multilingual Support with Unicode Datatypes**

An alternative to storing Unicode data in the database is to use the SQL NCHAR datatypes (NCHAR, NVARCHAR, NCLOB). You can store Unicode characters into
columns of these datatypes regardless of how the database character set has been defined. The NCHAR datatype is a Unicode datatype exclusively. In other words, it stores data encoded as Unicode.

In releases before Oracle9i, the NCHAR datatype supported fixed-width Asian character sets that were designed to provide higher performance. Examples of fixed-width character sets are JA16SJISFIXED and ZHT32EUCFIXED. No Unicode character set was supported as the national character set before Oracle9i.

You can create a table using the NVARCHAR2 and NCHAR datatypes. The column length specified for the NCHAR and NVARCHAR2 columns is always the number of characters instead of the number of bytes:

```
CREATE TABLE product_information
( product_id          NUMBER(6),
  product_name        NVARCHAR2(100),
  product_description VARCHAR2(1000));
```

The encoding used in the SQL NCHAR datatypes is the national character set specified for the database. You can specify one of the following Oracle character sets as the national character set:

- **AL16UTF16**
  
  This is the default character set for SQL NCHAR datatypes. The character set encodes Unicode data in the UTF-16 encoding. It supports supplementary characters, which are stored as four bytes.

- **UTF8**
  
  When UTF8 is specified for SQL NCHAR datatypes, the data stored in the SQL datatypes is in UTF-8 encoding.

You can specify the national character set for the SQL NCHAR datatypes when you create a database using the CREATE DATABASE statement with the NATIONAL CHARACTER SET clause. The following statement creates a database with WE8ISO8859P1 as the database character set and AL16UTF16 as the national character set.

```
Example 6–2 Creating a Database with a National Character Set

CREATE DATABASE sample
  CONTROLFILE REUSE
  LOGFILE
    GROUP 1 ('diskx:log1.log', 'disky:log1.log') SIZE 50K,
    GROUP 2 ('diskx:log2.log', 'disky:log2.log') SIZE 50K
```
Implementing a Unicode Solution in the Database

MAXLOGFILES 5
MAXLOGHISTORY 100
MAXDATAFILES 10
MAXINSTANCES 2
ARCHIVERLOG
CHARACTER SET WE8ISO8859P1
NATIONAL CHARACTER SET AL16UTF16
DATAFILE
  'disk1:df1.dbf' AUTOEXTEND ON,
  'disk2:df2.dbf' AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED
DEFAULT TEMPORARY TABLESPACE temp_ts
UNDO TABLESPACE undo_ts
SET TIME_ZONE = '+02:00';

How to Choose Between a Unicode Database and a Unicode Datatype Solution

To choose the right Unicode solution for your database, consider the following questions:

- Programming environment: What are the main programming languages used in your applications? How do they support Unicode?
- Ease of migration: How easily can your data and applications be migrated to take advantage of the Unicode solution?
- Performance: How much performance overhead are you willing to accept in order to use Unicode in the database?
- Type of data: Is your data mostly Asian or European? Do you need to store multilingual documents into LOB columns?
- Type of applications: What type of applications are you implementing: a packaged application or a customized end-user application?

This section describes some general guidelines for choosing a Unicode database or a Unicode datatype solution. The final decision largely depends on your exact environment and requirements. This section contains the following topics:

- When Should You Use a Unicode Database?
- When Should You Use Unicode Datatypes?

When Should You Use a Unicode Database?
Use a Unicode database in the situations described in Table 6–2.
Implementing a Unicode Solution in the Database

**Table 6-2 Using a Unicode Database**

<table>
<thead>
<tr>
<th>Situation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>You need easy code migration for Java or PL/SQL.</td>
<td>If your existing application is mainly written in Java and PL/SQL and your main concern is to minimize the code changes required to support multiple languages, then you may want to use a Unicode database solution. If the datatypes used to stored data remain as SQL CHAR datatypes, then the Java and PL/SQL code that accesses these columns does not need to change.</td>
</tr>
<tr>
<td>You have evenly distributed multilingual data.</td>
<td>If the multilingial data is evenly distributed in existing schema tables and you are not sure which tables contain multilingual data, then you should use a Unicode database because it does not require you to identify the kind of data that is stored in each column.</td>
</tr>
<tr>
<td>Your SQL statements and PL/SQL code contain Unicode data.</td>
<td>You must use a Unicode database. SQL statements and PL/SQL code are converted into the database character set before being processed. If the SQL statements and PL/SQL code contain characters that cannot be converted to the database character set, then those characters are lost. A common place to use Unicode data in a SQL statement is in a string literal.</td>
</tr>
<tr>
<td>You want to store multilingual documents in BLOB format and use Oracle Text for content searching.</td>
<td>You must use a Unicode database. The BLOB data is converted to the database character set before being indexed by Oracle Text. If your database character set is not UTF-8, then data are lost when the documents contain characters that cannot be converted to the database character set.</td>
</tr>
</tbody>
</table>

**When Should You Use Unicode Datatypes?**

Use Unicode datatypes in the situations described in Table 6-3.

**Table 6-3 Using Unicode Datatypes**

<table>
<thead>
<tr>
<th>Situation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>You want to add multilingual support incrementally.</td>
<td>If you want to add Unicode support to the existing database without migrating the character set, then consider using Unicode datatypes to store Unicode data. You can add columns of the SQL NCHAR datatypes to existing tables or new tables to support multiple languages incrementally.</td>
</tr>
<tr>
<td>You want to build a packaged application.</td>
<td>If you are building a packaged application to sell to customers, then you may want to build the application using SQL NCHAR datatypes. The SQL NCHAR datatype is a reliable Unicode datatype in which the data is always stored in Unicode, and the length of the data is always specified in UTF-16 code units. As a result, you need to test the application only once. The application will run on customer databases with any database character set.</td>
</tr>
</tbody>
</table>
Implementing a Unicode Solution in the Database

Comparing Unicode Character Sets for Database and Datatype Solutions

Oracle provides two solutions to store Unicode characters in the database: a Unicode database solution and a Unicode datatype solution. After you select the Unicode database solution, the Unicode datatype solution or a combination of both, determine the character set to be used in the Unicode database or the Unicode datatype.

Table 6–4 contains advantages and disadvantages of different character sets for a Unicode database solution. The Oracle character sets that can be Unicode database character sets are AL32UTF8, UTF8, and UTFE.
<table>
<thead>
<tr>
<th>Database Character Set</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
</table>
| AL32UTF8               | • Supplementary characters are stored in 4 bytes, there is no data conversion when supplementary characters are retrieved and inserted if the client setting is UTF-8.  
• The storage for supplementary characters requires less disk space in AL32UTF8 than in UTF8. | • You cannot specify the length of SQL CHAR types in number of UCS-2 code points for supplementary characters. Supplementary characters are treated as one code point rather than the standard two code points.  
• The binary order for SQL CHAR columns is different from the binary order of SQL NCHAR columns when the data consists of supplementary characters. As a result, CHAR columns and NCHAR columns do not always have the same sort for identical strings. |
| UTF8                   | • You can specify the length of SQL CHAR types in number of UCS-2 code points.  
• The binary order of the SQL CHAR columns is always the same as the binary order of the SQL NCHAR columns when the data consists of the same supplementary characters. As a result, CHAR columns and NCHAR columns have the same sort for identical strings. | • Supplementary characters are stored as 6 bytes instead of the 4 bytes defined by Unicode 3.2. As a result, Oracle has to convert data for supplementary characters if the client setting is UTF-8. |
| UTFE                   | • This is the only Unicode character set for the EBCDIC platform.  
• You can specify the length of SQL CHAR types in number of UCS-2 code points.  
• The binary order of the SQL CHAR columns is always the same as the binary order of the SQL NCHAR columns when the data consists of the same supplementary characters. As a result, CHAR columns and NCHAR columns have the same sort for identical strings. | • Supplementary characters are stored as 8 bytes (two 4-byte sequences) instead of the 5 bytes defined by the Unicode standard. As a result, Oracle has to convert data for those supplementary characters.  
• UTFE is not a standard encoding in the Unicode standard. As a result, clients requiring standard UTF-8 encoding must convert data from UTFE to the standard encoding when data is retrieved and inserted. |
Table 6-5 contains advantages and disadvantages of different character sets for a Unicode datatype solution. The Oracle character sets that can be national character sets are AL16UTF16 and UTF8. The default is AL16UTF16.

**Table 6-5  Character Set Advantages and Disadvantages for a Unicode Datatype Solution**

<table>
<thead>
<tr>
<th>National Character Set</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
</table>
| AL16UTF16              | ■ Asian data in AL16UTF16 is usually more compact than in UTF8. As a result, you save disk space and have less disk I/O when most of the multilingual data stored in the database is Asian data.  
■ It is usually faster to process strings encoded in the AL16UTF16 character set than strings encoded in UTF8 because Oracle processes most characters in an AL16UTF16 encoded string as fixed-width characters.  
■ The maximum length limits for the NCHAR and NVARCHAR2 columns are 1000 and 2000 characters, respectively. Because the data is fixed-width, the lengths are guaranteed. | ■ European ASCII data requires more disk space to store in AL16UTF16 than in UTF8. If most of your data is European data, then it uses more disk space than if it were UTF8 data.  
■ The maximum lengths for NCHAR and NVARCHAR2 are 1000 and 2000 characters, which is less than the lengths for NCHAR (2000) and NVARCHAR2 (4000) in UTF8. |
| UTF8                   | ■ European data in UTF8 is usually more compact than in AL16UTF16. As a result, you save disk space and have better response time when most of the multilingual data stored in the database is European data.  
■ The maximum lengths for the NCHAR and NVARCHAR2 columns are 2000 and 4000 characters respectively, which is more than those for NCHAR (1000) and NVARCHAR2 (2000) in AL16UTF16. Although the maximum lengths of the NCHAR and NVARCHAR2 columns are larger in UTF8, the actual storage size is still bound by the byte limits of 2000 and 4000 bytes, respectively. For example, you can store 4000 UTF8 characters in an NVARCHAR2 column if all the characters are single byte, but only 4000/3 characters if all the characters are three bytes. | ■ Asian data requires more disk space to store in UTF8 than in AL16UTF16. If most of your data is Asian data, then disk space usage is not less efficient than when the character set is AL16UTF16.  
■ Although you can specify larger length limits for NCHAR and NVARCHAR, you are not guaranteed to be able to insert the number of characters specified by these limits. This is because UTF8 allows variable-width characters.  
■ It is usually slower to process strings encoded in UTF8 than strings encoded in AL16UTF16 because UTF8 encoded strings consist of variable-width characters. |
Unicode Case Studies

This section describes typical scenarios for storing Unicode characters in an Oracle database:

- Example 6–3, "Unicode Solution with a Unicode Database"
- Example 6–4, "Unicode Solution with Unicode Datatypes"
- Example 6–5, "Unicode Solution with a Unicode Database and Unicode Datatypes"

**Example 6–3  Unicode Solution with a Unicode Database**

An American company running a Java application would like to add German and French support in the next release of the application. They would like to add Japanese support at a later time. The company currently has the following system configuration:

- The existing database has a database character set of US7ASCII.
- All character data in the existing database is composed of ASCII characters.
- PL/SQL stored procedures are used in the database.
- The database is around 300 GB.
- There is a nightly downtime of 4 hours.

In this case, a typical solution is to choose UTF8 for the database character set because of the following reasons:

- The database is very large and the scheduled downtime is short. Fast migration of the database to Unicode is vital. Because the database is in US7ASCII, the easiest and fastest way of enabling the database to support Unicode is to switch the database character set to UTF8 by issuing the `ALTER DATABASE` statement. No data conversion is required because US7ASCII is a subset of UTF8.
- Because most of the code is written in Java and PL/SQL, changing the database character set to UTF8 is unlikely to break existing code. Unicode support is automatically enabled in the application.
- Because the application supports French, German, and Japanese, there are few supplementary characters. Both AL32UTF8 and UTF8 are suitable.
**Example 6-4 Unicode Solution with Unicode Datatypes**

A European company that runs its applications mainly on Windows platforms wants to add new Windows applications written in Visual C/C++. The new applications will use the existing database to support Japanese and Chinese customer names. The company currently has the following system configuration:

- The existing database has a database character set of WE8ISO8859P1.
- All character data in the existing database is composed of Western European characters.
- The database is around 50 GB.

A typical solution is to take the following actions:

- Use `NCHAR` and `NVARCHAR2` datatypes to store Unicode characters.
- Keep WE8ISO8859P1 as the database character set.
- Use AL16UTF16 as the national character set.

The reasons for this solution are:

- Migrating the existing database to a Unicode database required data conversion because the database character set is WE8ISO8859P1 (a Latin-1 character set), which is not a subset of UTF8. As a result, there would be some overhead in converting the data to UTF8.
- The additional languages are supported in new applications only. They do not depend on the existing applications or schemas. It is simpler to use the Unicode datatype in the new schema and keep the existing schemas unchanged.
- Only customer name columns require Unicode support. Using a single `NCHAR` column meets the customer’s requirements without migrating the entire database.
- Because the languages to be supported are mostly Asian languages, AL16UTF16 should be used as the national character set so that disk space is used more efficiently.
- The lengths of the SQL `NCHAR` datatypes are defined as number of characters. This is the same as the way they are treated when using `wchar_t` strings in Windows C/C++ programs. This reduces programming complexity.
- Existing applications using the existing schemas are unaffected.
Example 6–5  Unicode Solution with a Unicode Database and Unicode Datatypes

A Japanese company wants to develop a new Java application. The company expects that the application will support as many languages as possible in the long run.

- In order to store documents as is, the company decided to use the BLOB datatype to store documents of multiple languages.
- The company may also want to generate UTF-8 XML documents from the relational data for business-to-business data exchange.
- The back-end has Windows applications written in C/C++ using ODBC to access the Oracle database.

In this case, the typical solution is to create a Unicode database using AL32UTF8 as the database character set and use the SQL NCHAR datatypes to store multilingual data. The national character set should be set to AL16UTF16. The reasons for this solution are as follows:

- When documents of different languages are stored BLOB format, Oracle Text requires the database character set to be one of the UTF-8 character sets. Because the applications may retrieve relational data as UTF-8 XML format (where supplementary characters are stored as four bytes), AL32UTF8 should be used as the database character set to avoid data conversion when UTF-8 data is retrieved or inserted.
- Because applications are new and written in both Java and Windows C/C++, the company should use the SQL NCHAR datatype for its relational data. Both Java and Windows support the UTF-16 character datatype, and the length of a character string is always measured in the number of characters.
- If most of the data is for Asian languages, then AL16UTF16 should be used with the SQL NCHAR datatypes because AL16UTF16 offers better performance and storage efficiency.

Designing Database Schemas to Support Multiple Languages

In addition to choosing a Unicode solution, the following issues should be taken into consideration when the database schema is designed to support multiple languages:

- Specifying Column Lengths for Multilingual Data
- Storing Data in Multiple Languages
Specifying Column Lengths for Multilingual Data

When you use NCHAR and NVARCHAR2 datatypes for storing multilingual data, the column size specified for a column is defined in number of characters. (The number of characters means the number of Unicode code units.) Table 6–6 shows the maximum size of the NCHAR and NVARCHAR2 datatypes for the AL16UTF16 and UTF8 national character sets.

<table>
<thead>
<tr>
<th>National Character Set</th>
<th>Maximum Column Size of NCHAR Datatype</th>
<th>Maximum Column Size of NVARCHAR2 Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL16UTF16</td>
<td>1000 characters</td>
<td>2000 characters</td>
</tr>
<tr>
<td>UTF8</td>
<td>2000 bytes</td>
<td>4000 bytes</td>
</tr>
</tbody>
</table>

When you use CHAR and VARCHAR2 datatypes for storing multilingual data, the maximum length specified for each column is, by default, in number of bytes. If the database needs to support Thai, Arabic, or multibyte languages such as Chinese and Japanese, then the maximum lengths of the CHAR, VARCHAR, and VARCHAR2 columns may need to be extended. This is because the number of bytes required to encode these languages in UTF8 or AL32UTF8 may be significantly larger than the number of bytes for encoding English and Western European languages. For example, one Thai character in the Thai character set requires 3 bytes in UTF8 or AL32UTF8. In addition, the maximum column lengths for CHAR, VARCHAR, and VARCHAR2 datatypes are 2000 bytes, 4000 bytes, and 4000 bytes respectively. If applications need to store more than 4000 bytes, then they should use the CLOB datatype.

Storing Data in Multiple Languages

The Unicode character set includes characters of most written languages around the world, but it does not contain information about the language to which a given character belongs. In other words, a character such as ä does not contain information about whether it is a French or German character. In order to provide information in the language a user desires, data stored in a Unicode database should accompany the language information to which the data belongs.
There are many ways for a database schema to relate data to a language. The following sections provide different approaches:

- **Store Language Information with the Data**
- **Select Translated Data Using Fine-Grained Access Control**

**Store Language Information with the Data**

For data such as product descriptions or product names, you can add a language column (`language_id`) of `CHAR` or `VARCHAR2` datatype to the product table to identify the language of the corresponding product information. This enables applications to retrieve the information in the desired language. The possible values for this language column are the 3-letter abbreviations of the valid `NLS_LANGUAGE` values of the database.

**See Also:** Appendix A, "Locale Data" for a list of `NLS_LANGUAGE` values and their abbreviations

You can also create a view to select the data of the current language. For example:

```
ALTER TABLE scott.product_information add (language_id VARCHAR2(50));

CREATE OR REPLACE VIEW product AS
  SELECT product_id, product_name
  FROM   product_information
  WHERE  language_id = sys_context('USERENV','LANG');
```

**Select Translated Data Using Fine-Grained Access Control**

Fine-grained access control enables you to limit the degree to which a user can view information in a table or view. Typically, this is done by appending a `WHERE` clause. When you add a `WHERE` clause as a fine-grained access policy to a table or view, Oracle automatically appends the `WHERE` clause to any SQL statements on the table at run time so that only those rows satisfying the `WHERE` clause can be accessed.

You can use this feature to avoid specifying the desired language of an user in the `WHERE` clause in every `SELECT` statement in your applications. The following `WHERE` clause limits the view of a table to the rows corresponding to the desired language of a user:

```
WHERE language_id = sys_context('userenv', 'LANG')
```

Specify this `WHERE` clause as a fine-grained access policy for `product_information` as follows:
create function func1 ( sch varchar2 , obj varchar2 )
return varchar2(100);
begin
return 'language_id = sys_context(''userenv'', ''LANG'')';
end
/

DBMS_RLS.ADD_POLICY ('scott', 'product_information', 'lang_policy', 'scott', 'func1', 'select');

Then any SELECT statement on the product_information table automatically appends the WHERE clause.

See Also: Oracle Database Application Developer’s Guide - Fundamentals for more information about fine-grained access control

Storing Documents in Multiple Languages in LOB Datatypes

You can store documents in multiple languages in CLOB, NCLOB, or BLOB datatypes and set up Oracle Text to enable content search for the documents.

Data in CLOB columns is stored in a format that is compatible with UCS-2 when the database character set is multibyte, such as UTF8 or AL32UTF8. This means that the storage space required for an English document doubles when the data is converted. Storage for an Asian language document in a CLOB column requires less storage space than the same document in a LONG column using UTF8, typically around 30% less, depending on the contents of the document.

Documents in NCLOB format are also stored in a proprietary format that is compatible with UCS-2 regardless of the database character set or national character set. The storage space requirement is the same as for CLOB data. Document contents are converted to UTF-16 when they are inserted into a NCLOB column. If you want to store multilingual documents in a non-Unicode database, then choose NCLOB. However, content search on NCLOB is not yet supported.

Documents in BLOB format are stored as they are. No data conversion occurs during insertion and retrieval. However, SQL string manipulation functions (such as LENGTH or SUBSTR) and collation functions (such as NLS_SORT and ORDER BY) cannot be applied to the BLOB datatype.

Table 6–7 lists the advantages and disadvantages of the CLOB, NCLOB, and BLOB datatypes when storing documents:
Creating Indexes for Searching Multilingual Document Contents

Oracle Text enables you to build indexes for content search on multilingual documents stored in CLOB format and BLOB format. It uses a language-specific lexer to parse the CLOB or BLOB data and produces a list of searchable keywords.

Create a multilexer to search multilingual documents. The multilexer chooses a language-specific lexer for each row, based on a language column. This section describe the high level steps to create indexes for documents in multiple languages. It contains the following topics:

- Creating Multilexers
- Creating Indexes for Documents Stored in the CLOB Datatype
- Creating Indexes for Documents Stored in the BLOB Datatype

See Also: *Oracle Text Reference*

Creating Multilexers

The first step in creating the multilexer is the creation of language-specific lexer preferences for each language supported. The following example creates English, German, and Japanese lexers with PL/SQL procedures:

---

Table 6–7  Comparison of LOB Datatypes for Document Storage

<table>
<thead>
<tr>
<th>Datatypes</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
</table>
| CLOB      | - Content search support  
           | - String manipulation support  | - Depends on database character set  
|           |            | - Data conversion is necessary for insertion  
|           |            | - Cannot store binary documents  |
| NCLOB     | - Independent of database character set  
           | - String manipulation support  | - No content search support  
|           |            | - Data conversion is necessary for insertion  
|           |            | - Cannot store binary documents  |
| BLOB      | - Independent of database character set  
           | - Content search support  | - No string manipulation support  
|           | - No data conversion, data stored as is  
|           | - Can store binary documents such as Microsoft Word or Microsoft Excel  | -  |

---
After the language-specific lexer preferences are created, they need to be gathered together under a single multilexer preference. First, create the multilexer preference, using the `MULTI_LEXER` object:

```sql
ctx_ddl.create_preference('global_lexer', 'multi_lexer');
```

Now add the language-specific lexers to the multilexer preference using the `add_sub_lexer` call:

```sql
ctx_ddl.add_sub_lexer('global_lexer', 'german', 'german_lexer');
ctx_ddl.add_sub_lexer('global_lexer', 'japanese', 'japanese_lexer');
ctx_ddl.add_sub_lexer('global_lexer', 'default', 'english_lexer');
```

This nominates the `german_lexer` preference to handle German documents, the `japanese_lexer` preference to handle Japanese documents, and the `english_lexer` preference to handle everything else, using `DEFAULT` as the language.

**Creating Indexes for Documents Stored in the CLOB Datatype**

The multilexer decides which lexer to use for each row based on a language column in the table. This is a character column that stores the language of the document in a text column. Use the Oracle language name to identify the language of a document in this column. For example, if you use the `CLOB` datatype to store your documents, then add the language column to the table where the documents are stored:

```sql
CREATE TABLE globaldoc
(doc_id    NUMBER       PRIMARY KEY,
language  VARCHAR2(30),
text      CLOB);
```

To create an index for this table, use the multilexer preference and specify the name of the language column:

```sql
CREATE INDEX globalx ON globaldoc(text)
indextype IS ctxsys.context
parameters ('lexer
global_lexer
language
In addition to the language column, the character set and format columns must be added in the table where the documents are stored. The character set column stores the character set of the documents using the Oracle character set names. The format column specifies whether a document is a text or binary document. For example, the `CREATE TABLE` statement can specify columns called `characterset` and `format`:

```sql
CREATE TABLE globaldoc (
  doc_id       NUMBER       PRIMARY KEY,
  language     VARCHAR2(30),
  characterset VARCHAR2(30),
  format       VARCHAR2(10),
  text         BLOB
);
```

You can put word-processing or spreadsheet documents into the table and specify `binary` in the `format` column. For documents in HTML, XML and text format, you can put them into the table and specify `text` in the `format` column.

Because there is a column in which to specify the character set, you can store text documents in different character sets.

When you create the index, specify the names of the format and character set columns:

```sql
CREATE INDEX globalx ON globaldoc(text)
  indextype is ctxsys.context
  parameters ('filter inso_filter
    lexer global_lexer
    language column language
    format  column format
    charset column characterset');
```

You can use the `charset_filter` if all documents are in text format. The `charset_filter` converts data from the character set specified in the `charset` column to the database character set.
This chapter describes how to use Oracle’s database access products with Unicode. It contains the following topics:

- Overview of Programming with Unicode
- SQL and PL/SQL Programming with Unicode
- OCI Programming with Unicode
- Pro*C/C++ Programming with Unicode
- JDBC Programming with Unicode
- ODBC and OLE DB Programming with Unicode
- XML Programming with Unicode
Overview of Programming with Unicode

Oracle offers several database access products for inserting and retrieving Unicode data. Oracle offers database access products for commonly used programming environments such as Java and C/C++. Data is transparently converted between the database and client programs, which ensures that client programs are independent of the database character set and national character set. In addition, client programs are sometimes even independent of the character datatype, such as NCHAR or CHAR, used in the database.

To avoid overloading the database server with data conversion operations, Oracle always tries to move them to the client side database access products. In a few cases, data must be converted in the database, which affects performance. This chapter discusses details of the data conversion paths.

Database Access Product Stack and Unicode

Oracle Corporation offers a comprehensive set of database access products that allow programs from different development environments to access Unicode data stored in the database. These products are listed in Table 7–1.

Table 7–1 Oracle Database Access Products

<table>
<thead>
<tr>
<th>Programming Environment</th>
<th>Oracle Database Access Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
<td>Oracle Call Interface (OCI)</td>
</tr>
<tr>
<td></td>
<td>Oracle Pro*C/C++</td>
</tr>
<tr>
<td></td>
<td>Oracle ODBC driver</td>
</tr>
<tr>
<td></td>
<td>Oracle Provider for OLE DB</td>
</tr>
<tr>
<td></td>
<td>Oracle Data Provider for .NET</td>
</tr>
<tr>
<td>Java</td>
<td>Oracle JDBC OCI or thin driver</td>
</tr>
<tr>
<td></td>
<td>Oracle server-side thin driver</td>
</tr>
<tr>
<td></td>
<td>Oracle server-side internal driver</td>
</tr>
<tr>
<td>PL/SQL</td>
<td>Oracle PL/SQL and SQL</td>
</tr>
<tr>
<td>Visual Basic/C#</td>
<td>Oracle ODBC driver</td>
</tr>
<tr>
<td></td>
<td>Oracle Provider for OLE DB</td>
</tr>
</tbody>
</table>

Figure 7–1 shows how the database access products can access the database.
The Oracle Call Interface (OCI) is the lowest level API that the rest of the client-side database access products use. It provides a flexible way for C/C++ programs to access Unicode data stored in SQL \texttt{CHAR} and \texttt{NCHAR} datatypes. Using OCI, you can programmatically specify the character set (UTF-8, UTF-16, and others) for the data to be inserted or retrieved. It accesses the database through Oracle Net.

Oracle Pro*C/C++ enables you to embed SQL and PL/SQL in your programs. It uses OCI’s Unicode capabilities to provide UTF-16 and UTF-8 data access for SQL \texttt{CHAR} and \texttt{NCHAR} datatypes.

The Oracle ODBC driver enables C/C++, Visual Basic, and VBScript programs running on Windows platforms to access Unicode data stored in SQL \texttt{CHAR} and \texttt{NCHAR} datatypes of the database. It provides UTF-16 data access by implementing the \texttt{SQLWCHAR} interface specified in the ODBC standard specification.

The Oracle Provider for OLE DB enables C/C++, Visual Basic, and VBScript programs running on Windows platforms to access Unicode data stored in SQL \texttt{CHAR} and \texttt{NCHAR} datatypes. It provides UTF-16 data access through wide string OLE DB datatypes.

The Oracle Data Provider for .NET enables programs running in any .NET programming environment on Windows platforms to access Unicode data stored in SQL \texttt{CHAR} and \texttt{NCHAR} datatypes. It provides UTF-16 data access through Unicode datatypes.
Oracle JDBC drivers are the primary Java programmatic interface for accessing an Oracle database. Oracle provides the following JDBC drivers:

- The JDBC OCI driver that is used by Java applications and requires the OCI library
- The JDBC thin driver, which is a pure Java driver that is primarily used by Java applets and supports the Oracle Net protocol over TCP/IP
- The JDBC server-side thin driver, a pure Java driver used inside Java stored procedures to connect to another Oracle server
- The JDBC server-side internal driver that is used inside the Oracle server to access the data in the database

All drivers support Unicode data access to SQL CHAR and NCHAR datatypes in the database.

The PL/SQL and SQL engines process PL/SQL programs and SQL statements on behalf of client-side programs such as OCI and server-side PL/SQL stored procedures. They allow PL/SQL programs to declare CHAR, VARCHAR2, NCHAR, and NVARCHAR2 variables and to access SQL CHAR and NCHAR datatypes in the database.

The following sections describe how each of the database access products supports Unicode data access to an Oracle database and offer examples for using those products:

- SQL and PL/SQL Programming with Unicode
- OCI Programming with Unicode
- Pro*C/C++ Programming with Unicode
- JDBC Programming with Unicode
- ODBC and OLE DB Programming with Unicode

### SQL and PL/SQL Programming with Unicode

SQL is the fundamental language with which all programs and users access data in an Oracle database either directly or indirectly. PL/SQL is a procedural language that combines the data manipulating power of SQL with the data processing power of procedural languages. Both SQL and PL/SQL can be embedded in other programming languages. This section describes Unicode-related features in SQL and PL/SQL that you can deploy for multilingual applications.
This section contains the following topics:

- SQL NCHAR Datatypes
- Implicit Datatype Conversion Between NCHAR and Other Datatypes
- Exception Handling for Data Loss During Datatype Conversion
- Rules for Implicit Datatype Conversion
- SQL Functions for Unicode Datatypes
- Other SQL Functions
- Unicode String Literals
- Using the UTL_FILE Package with NCHAR Data

See Also:

- Oracle Database SQL Reference
- PL/SQL User’s Guide and Reference

SQL NCHAR Datatypes

There are three SQL NCHAR datatypes:

- The NCHAR Datatype
- The NVARCHAR2 Datatype
- The NCLOB Datatype

The NCHAR Datatype

When you define a table column or a PL/SQL variable as the NCHAR datatype, the length is always specified as the number of characters. For example, the following statement creates a column with a maximum length of 30 characters:

```sql
CREATE TABLE table1 (column1 NCHAR(30));
```

The maximum number of bytes for the column is determined as follows:

```
maximum number of bytes = (maximum number of characters) x (maximum number of bytes for each character)
```

For example, if the national character set is UTF8, then the maximum byte length is 30 characters times 3 bytes for each character, or 90 bytes.
The national character set, which is used for all NCHAR datatypes, is defined when the database is created. The national character set can be either UTF8 or AL16UTF16. The default is AL16UTF16.

The maximum column size allowed is 2000 characters when the national character set is UTF8 and 1000 when it is AL16UTF16. The actual data is subject to the maximum byte limit of 2000. The two size constraints must be satisfied at the same time. In PL/SQL, the maximum length of NCHAR data is 32767 bytes. You can define an NCHAR variable of up to 32767 characters, but the actual data cannot exceed 32767 bytes. If you insert a value that is shorter than the column length, then Oracle pads the value with blanks to whichever length is smaller: maximum character length or maximum byte length.

Note: UTF8 may affect performance because it is a variable-width character set. Excessive blank padding of NCHAR fields decreases performance. Consider using the NVARCHAR datatype or changing to the AL16UTF16 character set for the NCHAR datatype.

The NVARCHAR2 Datatype

The NVARCHAR2 datatype specifies a variable length character string that uses the national character set. When you create a table with an NVARCHAR2 column, you specify the maximum number of characters for the column. Lengths for NVARCHAR2 are always in units of characters, just as for NCHAR. Oracle subsequently stores each value in the column exactly as you specify it, if the value does not exceed the column’s maximum length. Oracle does not pad the string value to the maximum length.

The maximum column size allowed is 4000 characters when the national character set is UTF8 and 2000 when it is AL16UTF16. The maximum length of an NVARCHAR2 column in bytes is 4000. Both the byte limit and the character limit must be met, so the maximum number of characters that is actually allowed in an NVARCHAR2 column is the number of characters that can be written in 4000 bytes.

In PL/SQL, the maximum length for an NVARCHAR2 variable is 32767 bytes. You can define NVARCHAR2 variables up to 32767 characters, but the actual data cannot exceed 32767 bytes.

The following statement creates a table with one NVARCHAR2 column whose maximum length in characters is 2000 and maximum length in bytes is 4000.

```
CREATE TABLE table2 (column2 NVARCHAR2(2000));
```
The NCLOB Datatype

NCLOB is a character large object containing Unicode characters, with a maximum size of 4 gigabytes. Unlike the BLOB datatype, the NCLOB datatype has full transactional support so that changes made through SQL, the DBMS_LOB package, or OCI participate fully in transactions. Manipulations of NCLOB value can be committed and rolled back. Note, however, that you cannot save an NCLOB locator in a PL/SQL or OCI variable in one transaction and then use it in another transaction or session.

NCLOB values are stored in the database in a format that is compatible with UCS-2, regardless of the national character set. Oracle translates the stored Unicode value to the character set requested on the client or on the server, which can be fixed-width or variable-width. When you insert data into an NCLOB column using a variable-width character set, Oracle converts the data into a format that is compatible with UCS-2 before storing it in the database.

See Also: Oracle Database Application Developer’s Guide - Large Objects for more information about the NCLOB datatype

Implicit Datatype Conversion Between NCHAR and Other Datatypes

Oracle supports implicit conversions between SQL NCHAR datatypes and other Oracle datatypes, such as CHAR, VARCHAR2, NUMBER, DATE, ROWID, and CLOB. Any implicit conversions for CHAR and VARCHAR2 datatypes are also supported for SQL NCHAR datatypes. You can use SQL NCHAR datatypes the same way as SQL CHAR datatypes.

Type conversions between SQL CHAR datatypes and SQL NCHAR datatypes may involve character set conversion when the database and national character sets are different. Padding with blanks may occur if the target data is either CHAR or NCHAR.

See Also: Oracle Database SQL Reference

Exception Handling for Data Loss During Datatype Conversion

Data loss can occur during datatype conversion when character set conversion is necessary. If a character in the source character set is not defined in the target character set, then a replacement character is used in its place. For example, if you try to insert NCHAR data into a regular CHAR column and the character data in NCHAR (Unicode) form cannot be converted to the database character set, then the character is replaced by a replacement character defined by the database character set. The NLS_NCHAR_CONV_EXCP initialization parameter controls the behavior of data loss during character type conversion. When this parameter is set to TRUE, any
SQL statements that result in data loss return an ORA-12713 error and the corresponding operation is stopped. When this parameter is set to FALSE, data loss is not reported and the unconvertible characters are replaced with replacement characters. The default value is TRUE. This parameter works for both implicit and explicit conversion.

In PL/SQL, when data loss occurs during conversion of SQL CHAR and NCHAR datatypes, the LOSSY_CHARSET_CONVERSION exception is raised for both implicit and explicit conversion.

### Rules for Implicit Datatype Conversion

In some cases, conversion between datatypes is possible in only one direction. In other cases, conversion in both directions is possible. Oracle defines a set of rules for conversion between datatypes. **Table 7–2** contains the rules for conversion between datatypes.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT/UPDATE statement</td>
<td>Values are converted to the datatype of the target database column.</td>
</tr>
<tr>
<td>SELECT INTO statement</td>
<td>Data from the database is converted to the datatype of the target variable.</td>
</tr>
<tr>
<td>Variable assignments</td>
<td>Values on the right of the equal sign are converted to the datatype of the target variable on the left of the equal sign.</td>
</tr>
<tr>
<td>Parameters in SQL and PL/SQL functions</td>
<td>CHAR, VARCHAR2, NCHAR, and NVARCHAR2 are loaded the same way. An argument with a CHAR, VARCHAR2, NCHAR or NVARCHAR2 datatype is compared to a formal parameter of any of the CHAR, VARCHAR2, NCHAR or NVARCHAR2 datatypes. If the argument and formal parameter datatypes do not match exactly, then implicit conversions are introduced when data is copied into the parameter on function entry and copied out to the argument on function exit.</td>
</tr>
<tr>
<td>Concatenation</td>
<td></td>
</tr>
<tr>
<td>SQL CHAR or NCHAR datatypes and NUMBER datatype</td>
<td>Character value is converted to NUMBER datatype</td>
</tr>
<tr>
<td>SQL CHAR or NCHAR datatypes and DATE datatype</td>
<td>Character value is converted to DATE datatype</td>
</tr>
</tbody>
</table>
### Table 7–2 Rules for Conversion Between Datatypes (Cont.)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL CHAR or NCHAR datatypes and ROWID datatype</td>
<td>Character datatypes are converted to ROWID datatype</td>
</tr>
<tr>
<td>SQL NCHAR and SQL CHAR datatypes</td>
<td>Character values are converted to NUMBER datatype</td>
</tr>
<tr>
<td>SQL CHAR or NCHAR datatypes and NUMBER datatype</td>
<td>Character values are converted to NUMBER datatype</td>
</tr>
<tr>
<td>SQL CHAR or NCHAR datatypes and DATE datatype</td>
<td>Character values are converted to DATE datatype</td>
</tr>
<tr>
<td>SQL CHAR or NCHAR datatypes and ROWID datatype</td>
<td>Character values are converted to ROWID datatype</td>
</tr>
<tr>
<td>SQL NCHAR datatypes and SQL CHAR datatypes</td>
<td>Comparisons between SQL NCHAR datatypes and SQL CHAR datatypes are more complex because they can be encoded in different character sets. When CHAR and VARCHAR2 values are compared, the CHAR values are converted to VARCHAR2 values. When NCHAR and NVARCHAR2 values are compared, the NCHAR values are converted to NVARCHAR2 values. When there is comparison between SQL NCHAR datatypes and SQL CHAR datatypes, character set conversion occurs if they are encoded in different character sets. The character set for SQL NCHAR datatypes is always Unicode and can be either UTF8 or AL16UTF16 encoding, which have the same character repertoires but are different encodings of the Unicode standard. SQL CHAR datatypes use the database character set, which can be any character set that Oracle supports. Unicode is a superset of any character set supported by Oracle, so SQL CHAR datatypes can always be converted to SQL NCHAR datatypes without data loss.</td>
</tr>
</tbody>
</table>

### SQL Functions for Unicode Datatypes

SQL NCHAR datatypes can be converted to and from SQL CHAR datatypes and other datatypes using explicit conversion functions. The examples in this section use the table created by the following statement:

```sql
CREATE TABLE customers
(id NUMBER, name NVARCHAR2(50), address NVARCHAR2(200), birthdate DATE);
```
Example 7–1  Populating the Customer Table Using the TO_NCHAR Function

The TO_NCHAR function converts the data at run time, while the N function converts the data at compilation time.

```sql
INSERT INTO customers VALUES (1000,
   TO_NCHAR('John Smith'),N'500 Oracle Parkway',sysdate);
```

Example 7–2  Selecting from the Customer Table Using the TO_CHAR Function

The following statement converts the values of name from characters in the national character set to characters in the database character set before selecting them according to the LIKE clause:

```sql
SELECT name FROM customers WHERE TO_CHAR(name) LIKE '%Sm%';
```

You should see the following output:

```
NAME
--------------------------------------
John Smith
```

Example 7–3  Selecting from the Customer Table Using the TO_DATE Function

Using the N function shows that either NCHAR or CHAR data can be passed as parameters for the TO_DATE function. The datatypes can mixed because they are converted at run time.

```sql
DECLARE
   ndatestring NVARCHAR2(20) := N'12-SEP-1975';
BEGIN
   SELECT name into ndstr FROM customers
   WHERE (birthdate)> TO_DATE(ndatestring, 'DD-MON-YYYY', N'NLS_DATE_LANGUAGE = AMERICAN');
END;
```

As demonstrated in Example 7–3, SQL NCHAR data can be passed to explicit conversion functions. SQL CHAR and NCHAR data can be mixed together when using multiple string parameters.

See Also:  Oracle Database SQL Reference for more information about explicit conversion functions for SQL NCHAR datatypes
Other SQL Functions

Most SQL functions can take arguments of SQL 
NCHAR datatypes as well as mixed character datatypes. The return datatype is based on the type of the first argument. If a non-string datatype like NUMBER or DATE is passed to these functions, then it is converted to VARCHAR2. The following examples use the customer table created in "SQL Functions for Unicode Datatypes" on page 7-9.

Example 7-4 INSTR Function

```sql
SELECT INSTR(name, N'Sm', 1, 1) FROM customers;
```

Example 7-5 CONCAT Function

```sql
SELECT CONCAT(name, id) FROM customers;
```

id is converted to NVARCHAR2 and then concatenated with name.

Example 7-6 RPAD Function

```sql
SELECT RPAD(name, 100, ' ') FROM customers;
```

The following output results:
```
RPAD(NAME,100,'')
------------------------------------------
John Smith
```

Space character ' ' is converted to the corresponding character in the NCHAR character set and then padded to the right of name until the total display length reaches 100.

See Also: Oracle Database SQL Reference

Unicode String Literals

You can input Unicode string literals in SQL and PL/SQL as follows:

- Put a prefix N before a string literal that is enclosed with single quote marks. This explicitly indicates that the following string literal is an NCHAR string literal. For example, N'12-SEP-1975' is an NCHAR string literal.

- Mark a string literal with single quote marks. Because Oracle supports implicit conversions to SQL NCHAR datatypes, a string literal is converted to a SQL NCHAR datatype wherever necessary.
Use the `NCHR(n)` SQL function, which returns a unit of character code in the national character set, which is AL16UTF16 or UTF8. The result of concatenating several `NCHR(n)` functions is `NVARCHAR2` data. In this way, you can bypass the client and server character set conversions and create an `NVARCHAR2` string directly. For example, `NCHR(32)` represents a blank character.

Because `NCHR(n)` is associated with the national character set, portability of the resulting value is limited to applications that run in the national character set. If this is a concern, then use the `UNISTR` function to remove portability limitations.

Use the `UNISTR('string')` SQL function. `UNISTR('string')` converts a string to the national character set. To ensure portability and to preserve data, include only ASCII characters and Unicode encoding in the following form: `\xxxx`, where `xxxx` is the hexadecimal value of a character code value in UTF-16 encoding format. For example, `UNISTR('G\0061ry')` represents 'Gary'. The ASCII characters are converted to the database character set and then to the national character set. The Unicode encoding is converted directly to the national character set.

The last two methods can be used to encode any Unicode string literals.

**Using the UTL_FILE Package with NCHAR Data**

The `UTL_FILE` package was enhanced in Oracle9i to handle Unicode national character set data. The following functions and procedures were added:

- **OPEN_NCHAR**
  This function opens a file in Unicode for input or output, with the maximum line size specified. With this function, you can read or write a text file in Unicode instead of in the database character set.

- **GET_LINE_NCHAR**
This procedure reads text from the open file identified by the file handle and places the text in the output buffer parameter. With this procedure, you can read a text file in Unicode instead of in the database character set.

- **PUT_NCHAR**
  This procedure writes the text string stored in the buffer parameter to the open file identified by the file handle. With this procedure, you can write a text file in Unicode instead of in the database character set.

- **PUT_LINE_NCHAR**
  This procedure writes the text string stored in the buffer parameter to the open file identified by the file handle. With this procedure, you can write a text file in Unicode instead of in the database character set.

- **PUTF_NCHAR**
  This procedure is a formatted PUT_NCHAR procedure. With this procedure, you can write a text file in Unicode instead of in the database character set.

**OCI Programming with Unicode**

OCI is the lowest-level API for accessing a database, so it offers the best possible performance. When using Unicode with OCI, consider these topics:

- **OCIEnvNlsCreate() Function for Unicode Programming**
- **OCI Unicode Code Conversion**
- **When the NLS_LANG Character Set is UTF8 or AL32UTF8 in OCI**
- **Binding and Defining SQL CHAR Datatypes in OCI**
- **Binding and Defining SQL NCHAR Datatypes in OCI**
- **Binding and Defining CLOB and NCLOB Unicode Data in OCI**

**See Also:** Chapter 10, "OCI Programming in a Global Environment"
OCIEnvNlsCreate() Function for Unicode Programming

The OCIEnvNlsCreate() function is used to specify a SQL CHAR character set and a SQL NCHAR character set when the OCI environment is created. It is an enhanced version of the OCIEnvCreate() function and has extended arguments for two character set IDs. The OCI_UTF16ID UTF-16 character set ID replaces the Unicode mode introduced in Oracle9i release 1 (9.0.1). For example:

OCIEnv *envhp;
status = OCIEnvNlsCreate((OCIEnv **)&envhp,
(ub4) 0,
(void *) 0,
(void **) 0,
(void *(*)(void *)) 0,
(void *(*)(void *)) 0,
(void (*)(void *)) 0,
(size_t) 0,
(void **) 0,
(ub2) OCI_UTF16ID, /* Metadata and SQL CHAR character set */
(ub2) OCI_UTF16ID /* SQL NCHAR character set */);

The Unicode mode, in which the OCI_UTF16 flag is used with the OCIEnvCreate() function, is deprecated.

When OCI_UTF16ID is specified for both SQL CHAR and SQL NCHAR character sets, all metadata and bound and defined data are encoded in UTF-16. Metadata includes SQL statements, user names, error messages, and column names. Thus, all inherited operations are independent of the NLS_LANG setting, and all metatext data parameters (text*) are assumed to be Unicode text datatypes (utext*) in UTF-16 encoding.

To prepare the SQL statement when the OCIEnv() function is initialized with the OCI_UTF16ID character set ID, call the OCISqlPrepare() function with a (utext*) string. The following example runs on the Windows platform only. You may need to change wchar_t datatypes for other platforms.

```c
const wchar_t sqlstr[] = L"SELECT * FROM ENAME=:ename";
...
OCISqlstmt* stmthp;
sts = OCHandleAlloc(envh, (void **) &stmthp, OCI_HTYPE_STMT, 0, NULL);
status = OCISqlPrepare(stmthp, errhp, (const text*)sqlstr, wcslen(sqlstr),
OCI_NTV_SYNTAX, OCI_DEFAULT);
```
To bind and define data, you do not have to set the `OCI_ATTR_CHARSET_ID` attribute because the `OCIEnv()` function has already been initialized with UTF-16 character set IDs. The bind variable names must also be UTF-16 strings.

```c
/* Inserting Unicode data */
OCIBindByName(stmthp1, &bnd1p, errhp, (const text*)L":ename",
(sb4)wcslen(L":ename"),
    (void *)ename, sizeof(ename), SQLT_STR, (void *)&insname_ind,
    (ub2 *) 0, (ub2 *) 0, (ub4) 0, (ub4 *)0,
OCI_DEFAULT);
OCIAttrSet((void *) bnd1p, (ub4) OCI_HTYPE_BIND, (void *)
    &ename_col_len,
    (ub4) 0, (ub4)OCI_ATTR_MAXDATA_SIZE, errhp);
...
/* Retrieving Unicode data */
OCIDefineByPos (stmthp2, &dfn1p, errhp, (ub4)1, (void *)ename,
    (sb4)sizeof(ename), SQLT_STR, (void *)0, (ub2 *)0,
    (ub2 *)0,
    (ub4)OCI_DEFAULT);
```

The `OCIExecute()` function performs the operation.

**See Also:** "Specifying Character Sets in OCI" on page 10-2

**OCI Unicode Code Conversion**

Unicode character set conversions take place between an OCI client and the database server if the client and server character sets are different. The conversion occurs on either the client or the server depending on the circumstances, but usually on the client side.

**Data Integrity**

You can lose data during conversion if you call an OCI API inappropriately. If the server and client character sets are different, then you can lose data when the destination character set is a smaller set than the source character set. You can avoid this potential problem if both character sets are Unicode character sets (for example, UTF8 and AL16UTF16).

When you bind or define SQL `NCHAR` datatypes, you should set the `OCI_ATTR_CHARSET_FORM` attribute to `SQLCS_NCHAR`. Otherwise, you can lose data because the data is converted to the database character set before converting to or from the national character set. This occurs only if the database character set is not Unicode.
OCI Performance Implications When Using Unicode

Redundant data conversions can cause performance degradation in your OCI applications. These conversions occur in two cases:

- When you bind or define SQL CHAR datatypes and set the `OCI_ATTR_CHARSET_FORM` attribute to `SQLCS_NCHAR`, data conversions take place from client character set to the national database character set, and from the national character set to the database character set. No data loss is expected, but two conversions happen, even though it requires only one.

- When you bind or define SQL NCHAR datatypes and do not set `OCI_ATTR_CHARSET_FORM`, data conversions take place from client character set to the database character set, and from the database character set to the national database character set. In the worst case, data loss can occur if the database character set is smaller than the client’s.

To avoid performance problems, you should always set `OCI_ATTR_CHARSET_FORM` correctly, based on the datatype of the target columns. If you do not know the target datatype, then you should set the `OCI_ATTR_CHARSET_FORM` attribute to `SQLCS_NCHAR` when binding and defining.

Table 7-3 contains information about OCI character set conversions.

<table>
<thead>
<tr>
<th>Datatypes for OCI Client Buffer</th>
<th><code>OCI_ATTR_CHARSET_FORM</code></th>
<th>Datatypes of the Target Column in the Database</th>
<th>Conversion Between</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>utext</td>
<td>SQLCS_IMPLICIT</td>
<td>CHAR, VARCHAR2, CLOB</td>
<td>UTF-16 and database character set in OCI</td>
<td>No unexpected data loss</td>
</tr>
<tr>
<td>utext</td>
<td>SQLCS_NCHAR</td>
<td>NCHAR, NVARCHAR2, NCLOB</td>
<td>UTF-16 and national character set in OCI</td>
<td>No unexpected data loss</td>
</tr>
<tr>
<td>utext</td>
<td>SQLCS_NCHAR</td>
<td>CHAR, VARCHAR2, CLOB</td>
<td>UTF-16 and national character set in OCI</td>
<td>No unexpected data loss, but may degrade performance because the conversion goes through the national character set</td>
</tr>
</tbody>
</table>

National character set and database character set in database server
OCI Programming with Unicode

Table 7–3  OCI Character Set Conversions (Cont.)

<table>
<thead>
<tr>
<th>Datatypes for OCI Client Buffer</th>
<th>OCI_ATTR_CHARSET_FORM</th>
<th>Datatypes of the Target Column in the Database</th>
<th>Conversion Between</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>utext</td>
<td>SQLCS_IMPLICIT</td>
<td>NCHAR, NVARCHAR2, NCLOB</td>
<td>UTF-16 and database character set in OCI Database character set and national character set in database server</td>
<td>Data loss may occur if the database character set is not Unicode</td>
</tr>
<tr>
<td>text</td>
<td>SQLCS_IMPLICIT</td>
<td>CHAR, VARCHAR2, CLOB</td>
<td>NLS_LANG character set and database character set in OCI</td>
<td>No unexpected data loss</td>
</tr>
<tr>
<td>text</td>
<td>SQLCS_NCHAR</td>
<td>NCHAR, NVARCHAR2, NCLOB</td>
<td>NLS_LANG character set and national character set in OCI</td>
<td>No unexpected data loss</td>
</tr>
<tr>
<td>text</td>
<td>SQLCS_NCHAR</td>
<td>CHAR, VARCHAR2, CLOB</td>
<td>NLS_LANG character set and national character set in OCI National character set and database character set in database server</td>
<td>No unexpected data loss, but may degrade performance because the conversion goes through the national character set</td>
</tr>
<tr>
<td>text</td>
<td>SQLCS_IMPLICIT</td>
<td>NCHAR, NVARCHAR2, NCLOB</td>
<td>NLS_LANG character set and database character set in OCI Database character set and national character set in database server</td>
<td>Data loss may occur because the conversion goes through the database character set</td>
</tr>
</tbody>
</table>

OCI Unicode Data Expansion

Data conversion can result in data expansion, which can cause a buffer to overflow. For binding operations, you need to set the OCI_ATTR_MAXDATA_SIZE attribute to a large enough size to hold the expanded data on the server. If this is difficult to do, then you need to consider changing the table schema. For defining operations, client applications need to allocate enough buffer space for the expanded data. The size of the buffer should be the maximum length of the expanded data. You can estimate the maximum buffer length with the following calculation:

1. Get the column data byte size.
2. Multiply it by the maximum number of bytes for each character in the client character set.
This method is the simplest and quickest way, but it may not be accurate and can waste memory. It is applicable to any character set combination. For example, for UTF-16 data binding and defining, the following example calculates the client buffer:

```c
ub2 csid = OCI_UTF16ID;
oratext *selstmt = "SELECT ename FROM emp";
counter = 1;
...
OCIStmtPrepare(stmthp, errhp, selstmt, (ub4)strlen((char*)selstmt),
                   OCI_NTV_SYNTAX, OCI_DEFAULT);
OCIStmtExecute ( svchp, stmthp, errhp, (ub4)0, (ub4)0,
                   (CONST OCISnapshot*)0, (OCISnapshot*)0,
                   OCI_DESCRIBE_ONLY);
OCIParamGet(stmthp, OCI_HTYPE_STMT, errhp, &myparam, (ub4)counter);
OCIAttrGet((void*)myparam, (ub4)OCI_DTYPE_PARAM, (void*)&col_width,
                   (ub4*)0, (ub4)OCI_ATTR_DATA_SIZE, errhp);
...
maxenamelen = (col_width + 1) * sizeof(utext);
cbuf = (utext*)malloc(maxenamelen);
...
OCIDefineByPos(stmthp, &dfnp, errhp, (ub4)1, (void *)cbuf,
                   (ub4)maxenamelen, SQLT_STR, (void *)0, (ub2 *)0,
                   (ub2*)0, (ub4)OCI_DEFAULT);
OCIAttrSet((void *) dfnp, (ub4) OCI_HTYPE_DEFINE, (void *)csid,
                   (ub4) 0, (ub4)OCI_ATTR_CHARSET_ID, errhp);
OCIStmtFetch(stmthp, errhp, 1, OCI_FETCH_NEXT, OCI_DEFAULT);
...
```

**When the NLS_LANG Character Set is UTF8 or AL32UTF8 in OCI**

You can use UTF8 and AL32UTF8 by setting NLS_LANG for OCI client applications. If you do not need supplementary characters, then it does not matter whether you choose UTF8 or AL32UTF8. However, if your OCI applications might handle supplementary characters, then you need to make a decision. Because UTF8 only supports characters of up to three bytes, no supplementary character can be represented in UTF8. In AL32UTF8, one supplementary character is represented in one code point, totalling four bytes.

Do not set NLS_LANG to AL16UTF16, because AL16UTF16 is the national character set for the server. If you need to use UTF-16, then you should specify the client character set to OCI_UTF16ID, using the OCIAttrSet() function when binding or defining data.
Binding and Defining SQL CHAR Datatypes in OCI

To specify a Unicode character set for binding and defining data with SQL CHAR datatypes, you may need to call the OCIAttrSet() function to set the appropriate character set ID after OCIBind() or OCIDefine() APIs. There are two typical cases:

- Call OCIBind() or OCIDefine() followed by OCIAttrSet() to specify UTF-16 Unicode character set encoding. For example:

```c
...  
ub2 csid = OCI_UTF16ID;  
utext ename[100]; /* enough buffer for ENAME */  
...  
/* Inserting Unicode data */  
OCIBindByName(stmthp1, &bnd1p, errhp, (oratext*)":ENAME",  
(sb4)strlen((char *)":ENAME"), (void *) ename, sizeof(ename),  
SQLT_STR, (void *)&insname_ind, (ub2 *) 0, (ub2 *) 0, (ub4) 0,  
(ub4 *)0, OCI_DEFAULT);  
OCIAttrSet((void *) bnd1p, (ub4) OCI_HTYPE_BIND, (void *) &csid,  
(ub4) 0, (ub4)OCI_ATTR_CHARSET_ID, errhp);  
OCIAttrSet((void *) bnd1p, (ub4) OCI_HTYPE_BIND, (void *) &ename_col_len,  
(ub4) 0, (ub4)OCI_ATTR_MAXDATA_SIZE, errhp);  
...  
/* Retrieving Unicode data */  
OCIDefineByPos (stmthp2, &dfn1p, errhp, (ub4)1, (void *)ename,  
(sb4)sizeof(ename), SQLT_STR, (void *)0, (ub2*)0,  
(ub2*)0, (ub4)OCI_DEFAULT);  
OCIAttrSet((void *) dfn1p, (ub4) OCI_HTYPE_DEFINE, (void *) &csid,  
(ub4) 0, (ub4)OCI_ATTR_CHARSET_ID, errhp);  
...  
```

If bound buffers are of the utext datatype, then you should add a cast (text*) when OCIBind() or OCIDefine() is called. The value of the OCI_ATTR_MAXDATA_SIZE attribute is usually determined by the column size of the server character set because this size is only used to allocate temporary buffer space for conversion on the server when you perform binding operations.

- Call OCIBind() or OCIDefine() with the NLS_LANG character set specified as UTF8 or AL32UTF8.

UTF8 or AL32UTF8 can be set in the NLS_LANG environment variable. You call OCIBind() and OCIDefine() in exactly the same manner as when you are not using Unicode. Set the NLS_LANG environment variable to UTF8 or AL32UTF8 and run the following OCI program:


oratext ename[100]; /* enough buffer size for ENAME */

/* Inserting Unicode data */
OCIBindByName(stmthp1, &bnd1p, errhp, (oratext*)":ENAME",
(sb4)strlen((char ")\"ENAME\")", (void *) ename, sizeof(ename),
SQLT_STR, (void *)&insname_ind, (ub2 *) 0, (ub2 *) 0,
(ub4) 0, (ub4 *)0, OCI_DEFAULT);
OCIAttrSet((void *) bnd1p, (ub4) OCI_HTYPE_BIND, (void *) &ename_col_len,
(ub4) 0, (ub4)OCI_ATTR_MAXDATA_SIZE, errhp);

/* Retrieving Unicode data */
OCIDefineByPos (stmthp2, &dfn1p, errhp, (ub4)1, (void *)ename,
(sb4)sizeof(ename), SQLT_STR, (void *)0, (ub2 *)0, (ub2*)0,
(ub4)OCI_DEFAULT);

---

**Binding and Defining SQL NCHAR Datatypes in OCI**

Oracle Corporation recommends that you access SQL NCHAR datatypes using UTF-16 binding or defining when using OCI. Beginning with Oracle9i, SQL NCHAR datatypes are Unicode datatypes with an encoding of either UTF8 or AL16UTF16. To access data in SQL NCHAR datatypes, set the OCI_ATTR_CHARSET_FORM attribute to SQLCS_NCHAR between binding or defining and execution so that it performs an appropriate data conversion without data loss. The length of data in SQL NCHAR datatypes is always in the number of Unicode code units.

The following program is a typical example of inserting and fetching data against an NCHAR data column:

...  
ub2 csid = OCI_UTF16ID;  
ub1 cform = SQLCS_NCHAR;  
utext ename[100]; /* enough buffer for ENAME */

/* Inserting Unicode data */
OCIBindByName(stmthp1, &bnd1p, errhp, (oratext*)":ENAME",
(sb4)strlen((char ")\"ENAME\")", (void *) ename,
sizeof(ename), SQLT_STR, (void *)&insname_ind, (ub2 *) 0,
(ub2 *) 0, (ub4) 0, (ub4 *)0, OCI_DEFAULT);
OCIAttrSet((void *) bnd1p, (ub4) OCI_HTYPE_BIND, (void *) &ename_col_len,
(ub4) 0, (ub4)OCI_ATTR_MAXDATA_SIZE, errhp);

OCIAttrSet((void *) bnd1p, (ub4) OCI_HTYPE_BIND, (void *) &cform, (ub4) 0,
(ub4)OCI_ATTR_CHARSET_FORM, errhp);
OCIAttrSet((void *) bnd1p, (ub4) OCI_HTYPE_BIND, (void *) &csid, (ub4) 0,
(ub4)OCI_ATTR_CHARSET_ID, errhp);
OCIAttrSet((void *) bnd1p, (ub4) OCI_HTYPE_BIND, (void *) &ename_col_len,
(ub4) 0, (ub4)OCI_ATTR_MAXDATA_SIZE, errhp);

/* Retrieving Unicode data */
OCIDefineByPos (stmthp2, &dfn1p, errhp, (ub4)1, (void *)ename,
   (sb4)sizeof(ename), SQLT_STR, (void *)0, (ub2)0, (ub2)0,
   (ub4)OCI_DEFAULT);
OCIAttrSet((void *) dfn1p, (ub4) OCI_HTYPE_DEFINE, (void *) &csid, (ub4) 0,
   (ub4)OCI_ATTR_CHARSET_ID, errhp);
OCIAttrSet((void *) dfn1p, (ub4) OCI_HTYPEDEFINE, (void *) &csid, (ub4) 0,
   (ub4)OCI_ATTR_CHARSET_ID, errhp);
OCIAttrSet((void *) dfn1p, (ub4) OCI_HTYPE_DEFINE, (void *) &cform, (ub4) 0,
   (ub4)OCI_ATTR_CHARSET_FORM, errhp);

Binding and Defining CLOB and NCLOB Unicode Data in OCI

In order to write (bind) and read (define) UTF-16 data for CLOB or NCLOB columns,
the UTF-16 character set ID must be specified as OCILobWrite() and
OCILobRead(). When you write UTF-16 data into a CLOB column, call
OCILobWrite() as follows:

...  
ub2 csid = OCI_UTF16ID;
err = OCILobWrite (ctx->svchp, ctx->errhp, lobp, &amtp, offset, (void *) buf,
   (ub4) BUFSIZE, OCI_ONE_PIECE, (void *)0,
   (sb4 (*)(void *)) 0, (ub2) csid, (ub1) SQLCS_IMPLICIT);

The amtp parameter is the data length in number of Unicode code units. The
offset parameter indicates the offset of data from the beginning of the data
column. The csid parameter must be set for UTF-16 data.

To read UTF-16 data from CLOB columns, call OCILobRead() as follows:

...  
ub2 csid = OCI_UTF16ID;
err = OCILobRead(ctx->svchp, ctx->errhp, lobp, &amtp, offset, (void *) buf,
   (ub4)BUFSIZE , (void *) 0, (sb4 (*)(void *)) 0, (ub2)csid,
   (ub1) SQLCS_IMPLICIT);

The data length is always represented in the number of Unicode code units. Note
one Unicode supplementary character is counted as two code units, because the
encoding is UTF-16. After binding or defining a LOB column, you can measure the
data length stored in the LOB column using OCILobGetLength(). The returning
value is the data length in the number of code units if you bind or define as UTF-16.

err = OCILobGetLength(ctx->svchp, ctx->errhp, lobp, &lenp);
If you are using an NCLOB, then you must set OCI_ATTR_CHARSET_FORM to SQLCS_NCHAR.

**Pro*C/C++ Programming with Unicode**

Pro*C/C++ provides the following ways to insert or retrieve Unicode data into or from the database:

- Using the VARCHAR Pro*C/C++ datatype or the native C/C++ text datatype, a program can access Unicode data stored in SQL CHAR datatypes of a UTF8 or AL32UTF8 database. Alternatively, a program could use the C/C++ native text type.
- Using the UVARCHAR Pro*C/C++ datatype or the native C/C++ utext datatype, a program can access Unicode data stored in NCHAR datatypes of a database.
- Using the NVARCHAR Pro*C/C++ datatype, a program can access Unicode data stored in NCHAR datatypes. The difference between UVARCHAR and NVARCHAR in a Pro*C/C++ program is that the data for the UVARCHAR datatype is stored in a utext buffer while the data for the NVARCHAR datatype is stored in a text datatype.

Pro*C/C++ does not use the Unicode OCI API for SQL text. As a result, embedded SQL text must be encoded in the character set specified in the NLS_LANG environment variable.

This section contains the following topics:

- **Pro*C/C++ Data Conversion in Unicode**
- **Using the VARCHAR Datatype in Pro*C/C++**
- **Using the NVARCHAR Datatype in Pro*C/C++**
- **Using the UVARCHAR Datatype in Pro*C/C++**

**Pro*C/C++ Data Conversion in Unicode**

Data conversion occurs in the OCI layer, but it is the Pro*C/C++ preprocessor that instructs OCI which conversion path should be taken based on the datatypes used in a Pro*C/C++ program. Table 7–4 illustrates the conversion paths:
Using the VARCHAR Datatype in Pro*C/C++

The Pro*C/C++ VARCHAR datatype is preprocessed to a struct with a length field and text buffer field. The following example uses the C/C++ text native datatype and the VARCHAR Pro*C/C++ datatypes to bind and define table columns.

```c
#include <sqlca.h>
main()
{
    ...
    /* Change to STRING datatype: */
    EXEC ORACLE OPTION (CHAR_MAP=STRING) ;
    text ename[20] ; /* unsigned short type */
    varchar address[50] ; /* Pro*C/C++ varchar type */
    
    EXEC SQL SELECT ename, address INTO :ename, :address FROM emp;
    /* ename is NULL-terminated */
    printf(L"ENAME = %s, ADDRESS = %.*s\n", ename, address.len, address.arr);
    ...
```
When you use the VARCHAR datatype or native text datatype in a Pro*C/C++ program, the preprocessor assumes that the program intends to access columns of SQL CHAR datatypes instead of SQL NCHAR datatypes in the database. The preprocessor generates C/C++ code to reflect this fact by doing a bind or define using the SQLCS_IMPLICIT value for the OCI_ATTR_CHARSET_FORM attribute. As a result, if a bind or define variable is bound to a column of SQL NCHAR datatypes in the database, then implicit conversion occurs in the database server to convert the data from the database character set to the national database character set and vice versa. During the conversion, data loss occurs when the database character set is a smaller set than the national character set.

Using the NVARCHAR Datatype in Pro*C/C++

The Pro*C/C++ NVARCHAR datatype is similar to the Pro*C/C++ VARCHAR datatype. It should be used to access SQL NCHAR datatypes in the database. It tells Pro*C/C++ preprocessor to bind or define a text buffer to the column of SQL NCHAR datatypes. The preprocessor specifies the SQLCS_NCHAR value for the OCI_ATTR_CHARSET_FORM attribute of the bind or define variable. As a result, no implicit conversion occurs in the database.

If the NVARCHAR buffer is bound against columns of SQL CHAR datatypes, then the data in the NVARCHAR buffer (encoded in the NLS_LANG character set) is converted to or from the national character set in OCI, and the data is then converted to the database character set in the database server. Data can be lost when the NLS_LANG character set is a larger set than the database character set.

Using the UVARCHAR Datatype in Pro*C/C++

The UVARCHAR datatype is preprocessed to a struct with a length field and utext buffer field. The following example code contains two host variables, ename and address. The ename host variable is declared as a utext buffer containing 20 Unicode characters. The address host variable is declared as a uvarchar buffer containing 50 Unicode characters. The len and arr fields are accessible as fields of a struct.

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

main()
{
...
```
When you use the UVARCHAR datatype or native utext datatype in Pro*C/C++ programs, the preprocessor assumes that the program intends to access SQL NCHAR datatypes. The preprocessor generates C/C++ code by binding or defining using the SQLCS_NCHAR value for OCI_ATTR_CHARSET_FORM attribute. As a result, if a bind or define variable is bound to a column of a SQL NCHAR datatype, then an implicit conversion of the data from the national character set occurs in the database server. However, there is no data lost in this scenario because the national character set is always a larger set than the database character set.

**JDBC Programming with Unicode**

Oracle provides the following JDBC drivers for Java programs to access character data in an Oracle database:

- The JDBC OCI driver
- The JDBC thin driver
- The JDBC server-side internal driver
- The JDBC server-side thin driver

Java programs can insert or retrieve character data to and from columns of SQL CHAR and NCHAR datatypes. Specifically, JDBC enables Java programs to bind or define Java strings to SQL CHAR and NCHAR datatypes. Because Java's string datatype is UTF-16 encoded, data retrieved from or inserted into the database must be converted from UTF-16 to the database character set or the national character set and vice versa. JDBC also enables you to specify the PL/SQL and SQL statements in Java strings so that any non-ASCII schema object names and string literals can be used.

At database connection time, JDBC sets the server NLS_LANGUAGE and NLS_TERRITORY parameters to correspond to the locale of the Java VM that runs the
JDBC driver. This operation ensures that the server and the Java client communicate in the same language. As a result, Oracle error messages returned from the server are in the same language as the client locale.

This section contains the following topics:

- Binding and Defining Java Strings to SQL CHAR Datatypes
- Binding and Defining Java Strings to SQL NCHAR Datatypes
- Using the SQL NCHAR Datatypes Without Changing the Code
- Data Conversion in JDBC
- Using oracle.sql.CHAR in Oracle Object Types
- Restrictions on Accessing SQL CHAR Data with JDBC

### Binding and Defining Java Strings to SQL CHAR Datatypes

Oracle JDBC drivers allow you to access SQL CHAR datatypes in the database using Java string bind or define variables. The following code illustrates how to bind a Java string to a CHAR column.

```java
int employee_id = 12345;
String last_name = "Joe";
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO" +
   "employees (last_name, employee_id) VALUES (?, ?)");
pstmt.setString(1, last_name);
pstmt.setInt(2, employee_id);
pstmt.execute(); /* execute to insert into first row */
employee_id += 1; /* next employee number */
last_name = "\uFF2A\uFF4F\uFF45"; /* Unicode characters in name */
pstmt.setString(1, last_name);
pstmt.setInt(2, employee_id);
pstmt.execute(); /* execute to insert into second row */
```

You can define the target SQL columns by specifying their datatypes and lengths. When you define a SQL CHAR column with the datatype and the length, JDBC uses this information to optimize the performance of fetching SQL CHAR data from the column. The following is an example of defining a SQL CHAR column.

```java
OraclePreparedStatement pstmt = (OraclePreparedStatement)
   conn.prepareStatement("SELECT ename, empno from emp");
pstmt.defineColumnType(1, Types.VARCHAR, 3);
pstmt.defineColumnType(2, Types.INTEGER);
ResultSet rest = pstmt.executeQuery();
```
String name = rset.getString(1);
int id = reset.getInt(2);

You need to cast PreparedStatement to OraclePreparedStatement to call defineColumnType(). The second parameter of defineColumnType() is the datatype of the target SQL column. The third parameter is the length in number of characters.

### Binding and Defining Java Strings to SQL NCHAR Datatypes

For binding or defining Java string variables to SQL NCHAR datatypes, Oracle provides an extended PreparedStatement which has the setFormOfUse() method through which you can explicitly specify the target column of a bind variable to be a SQL NCHAR datatype. The following code illustrates how to bind a Java string to an NCHAR column.

```java
int employee_id = 12345;
String last_name = "Joe"
oracle.jdbc.OraclePreparedStatement pstmt =
    (oracle.jdbc.OraclePreparedStatement)
    conn.prepareStatement("INSERT INTO employees (last_name, employee_id)
VALUES (?, ?)");
pstmt.setFormOfUse(1, oracle.jdbc.OraclePreparedStatement.FORM_NCHAR);
pstmt.setString(1, last_name);
pstmt.setInt(2, employee_id);
pstmt.execute();                 /* execute to insert into first row */
employee_id += 1;                      /* next employee number */
last_name = "\uFF2A\uFF4F\uFF45";    /* Unicode characters in name */
pstmt.setString(1, last_name);
pstmt.setInt(2, employee_id);
pstmt.execute();                 /* execute to insert into second row */
```

You can define the target SQL NCHAR columns by specifying their datatypes, forms of use, and lengths. JDBC uses this information to optimize the performance of fetching SQL NCHAR data from these columns. The following is an example of defining a SQL NCHAR column.

```java
OraclePreparedStatement pstmt = (OraclePreparedStatement)
    conn.prepareStatement("SELECT ename, empno from emp");
pstmt.defineColumnType(1, Types.VARCHAR, 3,
OraclePreparedStatement.FORM_NCHAR);
pstmt.defineColumnType(2, Types.INTEGER);
ResultSet rest = pstmt.executeQuery();
String name = rset.getString(1);
```
int id = reset.getInt(2);

To define a SQL NCHAR column, you need to specify the datatype that is equivalent to a SQL CHAR column in the first argument, the length in number of characters in the second argument, and the form of use in the fourth argument of defineColumnType().

You can bind or define a Java string against an NCHAR column without explicitly specifying the form of use argument. This implies the following:

- If you do not specify the argument in the setString() method, then JDBC assumes that the bind or define variable is for the SQL CHAR column. As a result, it tries to convert them to the database character set. When the data gets to the database, the database implicitly converts the data in the database character set to the national character set. During this conversion, data can be lost when the database character set is a subset of the national character set. Because the national character set is either UTF8 or AL16UTF16, data loss would happen if the database character set is not UTF8 or AL32UTF8.

- Because implicit conversion from SQL CHAR to SQL NCHAR datatypes happens in the database, database performance is degraded.

In addition, if you bind or define a Java string for a column of SQL CHAR datatypes but specify the form of use argument, then performance of the database is degraded. However, data should not be lost because the national character set is always a larger set than the database character set.

Using the SQL NCHAR Datatypes Without Changing the Code

A global flag has been introduced in the Oracle JDBC drivers for customers to tell whether the form of use argument should be specified by default in a Java application. This flag has the following purposes:

- Existing applications accessing the SQL CHAR datatypes can be migrated to support the SQL NCHAR datatypes for worldwide deployment without changing a line of code.

- Applications do not need to call the setFormOfUse() method when binding and defining a SQL NCHAR column. The application code can be made neutral and independent of the datatypes being used in the backend database. With this flag, applications can be easily switched from using SQL CHAR or SQL NCHAR.

The global flag is specified in the command line that invokes the Java application. The syntax of specifying this flag is as follows:
java -Doracle.jdbc.defaultNChar=true <application class>

With this flag specified, the Oracle JDBC drivers assume the presence of the form of use argument for all bind and define operations in the application.

If you have a database schema that consists of both the SQL CHAR and SQL NCHAR columns, then using this flag may have some performance impact when accessing the SQL CHAR columns because of implicit conversion done in the database server.

See Also: "Data Conversion in JDBC" on page 7-29 for more information about the performance impact of implicit conversion

Data Conversion in JDBC

Because Java strings are always encoded in UTF-16, JDBC drivers transparently convert data from the database character set to UTF-16 or the national character set. The conversion paths taken are different for the JDBC drivers:

- Data Conversion for the OCI Driver
- Data Conversion for Thin Drivers
- Data Conversion for the Server-Side Internal Driver

Data Conversion for the OCI Driver

For the OCI driver, the SQL statements are always converted to the database character set by the driver before it is sent to the database for processing. When the database character set is neither US7ASCII nor WE8ISO8859P1, the driver converts the SQL statements to UTF-8 first in Java and then to the database character set in C. Otherwise, it converts the SQL statements directly to the database character set. For Java string bind or define variables, Table 7–5 summarizes the conversion paths taken for different scenarios.

<table>
<thead>
<tr>
<th>Form of Use</th>
<th>SQL Datatype</th>
<th>Conversion Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Const.CHAR</td>
<td>CHAR</td>
<td>Java string to and from database character set happens in the JDBC driver</td>
</tr>
<tr>
<td>(Default)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Const.CHAR</td>
<td>NCHAR</td>
<td>Java string to and from database character set happens in the JDBC driver. Data in the database character set to and from national character set happens in the database server</td>
</tr>
<tr>
<td>(Default)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SQL statements are always converted to either the database character set or to UTF-8 by the driver before they are sent to the database for processing. When the database character set is either US7ASCII or WE8ISO8859P1, the driver converts the SQL statement to the database character set. Otherwise, the driver converts the SQL statement to UTF-8 and notifies the database that a SQL statement requires further conversion before being processed. The database, in turn, converts the SQL statements from UTF-8 to the database character set. The database, in turn, converts the SQL statement to the database character set. For Java string bind and define variables, the conversion paths shown in Table 7–6 are taken for the thin driver.

### Table 7–6 Thin Driver Conversion Path

<table>
<thead>
<tr>
<th>Form of Use</th>
<th>SQL Datatype</th>
<th>Database Character Set</th>
<th>Conversion Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Const.CHAR</td>
<td>CHAR</td>
<td>US7ASCII or WE8ISO8859P1</td>
<td>Java string to and from UTF-8 happens in the thin driver</td>
</tr>
<tr>
<td>Const.CHAR</td>
<td>NCHAR</td>
<td>US7ASCII or WE8ISO8859P1</td>
<td>Data in UTF-8 to and from the database character set happens in the database server</td>
</tr>
<tr>
<td>Const.CHAR</td>
<td>CHAR</td>
<td>non-ASCII and non-WE8ISO8859P1</td>
<td>Java string to and from UTF-8 happens in the thin driver</td>
</tr>
<tr>
<td>Const.CHAR</td>
<td>NCHAR</td>
<td>non-ASCII and non-WE8ISO8859P1</td>
<td>Data in UTF-8 to and from national character set happens in the database server</td>
</tr>
</tbody>
</table>
Data Conversion for the Server-Side Internal Driver

All data conversion occurs in the database server because the server-side internal driver works inside the database.

Using oracle.sql.CHAR in Oracle Object Types

JDBC drivers support Oracle object types. Oracle objects are always sent from database to client as an object represented in the database character set or national character set. That means the data conversion path in "Data Conversion in JDBC" on page 7-29 does not apply to Oracle object access. Instead, the oracle.sql.CHAR class is used for passing SQL CHAR and SQL NCHAR data of an object type from the database to the client.

This section includes the following topics:

- oracle.sql.CHAR
- Accessing SQL CHAR and NCHAR Attributes with oracle.sql.CHAR

oracle.sql.CHAR

The oracle.sql.CHAR class has a special functionality for conversion of character data. The Oracle character set is a key attribute of the oracle.sql.CHAR class. The Oracle character set is always passed in when an oracle.sql.CHAR object is constructed. Without a known character set, the bytes of data in the oracle.sql.CHAR object are meaningless.

The oracle.sql.CHAR class provides the following methods for converting character data to strings:

- getString()
Converts the sequence of characters represented by the `oracle.sql.CHAR` object to a string, returning a Java string object. If the character set is not recognized, then `getString()` returns a `SQLException`.

- `toString()`:
  Identical to `getString()`, except that if the character set is not recognized, then `toString()` returns a hexadecimal representation of the `oracle.sql.CHAR` data and does not returns a `SQLException`.

- `getStringWithReplacement()`:
  Identical to `getString()`, except that a default replacement character replaces characters that have no Unicode representation in the character set of this `oracle.sql.CHAR` object. This default character varies among character sets, but it is often a question mark.

You may want to construct an `oracle.sql.CHAR` object yourself (to pass into a prepared statement, for example). When you construct an `oracle.sql.CHAR` object, you must provide character set information to the `oracle.sql.CHAR` object by using an instance of the `oracle.sql.CharacterSet` class. Each instance of the `oracle.sql.CharacterSet` class represents one of the character sets that Oracle supports.

Complete the following tasks to construct an `oracle.sql.CHAR` object:

1. Create a `CharacterSet` instance by calling the static `CharacterSet.make()` method. This method creates the character set class. It requires as input a valid Oracle character set (OracleId). For example:

   ```java
   int OracleId = CharacterSet.JA16SJIS_CHARSET; // this is character set 832
   ...  
   CharacterSet mycharset = CharacterSet.make(OracleId);
   ```

   Each character set that Oracle supports has a unique predefined `OracleId`. The `OracleId` can always be referenced as a character set specified as `Oracle_character_set_name_CHARSET` where `Oracle_character_set_name` is the Oracle character set.

2. Construct an `oracle.sql.CHAR` object. Pass to the constructor a string (or the bytes that represent the string) and the `CharacterSet` object that indicates how to interpret the bytes based on the character set. For example:

   ```java
   String mystring = "teststring";
   ...  
   oracle.sql.CHAR mychar = new oracle.sql.CHAR(teststring, mycharset);
   ```
The `oracle.sql.CHAR` class has multiple constructors: they can take a string, a byte array, or an object as input along with the `CharacterSet` object. In the case of a string, the string is converted to the character set indicated by the `CharacterSet` object before being placed into the `oracle.sql.CHAR` object.

The server (database) and the client (or application running on the client) can use different character sets. When you use the methods of this class to transfer data between the server and the client, the JDBC drivers must convert the data between the server character set and the client character set.

**Accessing SQL CHAR and NCHAR Attributes with `oracle.sql.CHAR`**

The following is an example of an object type created using SQL:

```sql
CREATE TYPE person_type AS OBJECT (name VARCHAR2(30), address NVARCHAR(256), age NUMBER);
CREATE TABLE employees (id NUMBER, person PERSON_TYPE);
```

The Java class corresponding to this object type can be constructed as follows:

```java
public class person implements SqlData {
    oracle.sql.CHAR name;
    oracle.sql.CHAR address;
    oracle.sql.NUMBER age;
    // SqlData interfaces
    getSqlType() {...}
    writeSql(SqlOutput stream) {...}
    readSql(SqlInput stream, String sqltype) {...}
}
```

The `oracle.sql.CHAR` class is used here to map to the `NAME` attributes of the Oracle object type, which is of VARCHAR2 datatype. JDBC populates this class with the byte representation of the VARCHAR2 data in the database and the `CharacterSet` object corresponding to the database character set. The following code retrieves a `person` object from the `employees` table:

```java
TypeMap map = ((OracleConnection)conn).getTypeMap();
map.put("PERSON_TYPE", Class.forName("person"));
conn.setTypeMap(map);

ResultSet rs = stmt.executeQuery("SELECT PERSON FROM EMPLOYEES");
rs.next();
person p = (person) rs.getObject(1);
```
oracle.sql.CHAR sql_name = p.name;
oracle.sql.CHAR sql_address=p.address;
String java_name = sql_name.getString();
String java_address = sql_address.getString();

The `getString()` method of the `oracle.sql.CHAR` class converts the byte array from the database character set or national character set to UTF-16 by calling Oracle's Java data conversion classes and returning a Java string. For the `rs.getObject(1)` call to work, the `SqlData` interface has to be implemented in the class `person`, and the `Typemap` map has to be set up to indicate the mapping of the object type `PERSON_TYPE` to the Java class.

**Restrictions on Accessing SQL CHAR Data with JDBC**

This section contains the following topics:

- SQL CHAR Data Size Restriction With the JDBC Thin Driver
- Character Integrity Issues in a Multibyte Database Environment

**SQL CHAR Data Size Restriction With the JDBC Thin Driver**

If the database character set is neither ASCII (US7ASCII) nor ISO Latin1 (WE8ISO8859P1), then the JDBC thin driver must impose size restrictions for SQL CHAR bind parameters that are more restrictive than normal database size limitations. This is necessary to allow for data expansion during conversion.

The JDBC thin driver checks SQL CHAR bind sizes when a `setXXX()` method (except for the `setCharacterStream()` method) is called. If the data size exceeds the size restriction, then the driver returns a SQL exception (`SQLException: Data size bigger than max size for this type`) from the `setXXX()` call. This limitation is necessary to avoid the chance of data corruption when conversion of character data occurs and increases the length of the data. This limitation is enforced in the following situations:

- Using the JDBC thin driver
- Using binds (not defines)
- Using SQL CHAR datatypes
- Connecting to a database whose character set is neither ASCII (US7ASCII) nor ISO Latin1 (WE8ISO8859P1)

When the database character set is neither US7ASCII nor WE8ISO8859P1, the JDBC thin driver converts Java UTF-16 characters to UTF-8 encoding bytes for SQL CHAR...
binds. The UTF-8 encoding bytes are then transferred to the database, and the database converts the UTF-8 encoding bytes to the database character set encoding. This conversion to the character set encoding can result in an increase in the number of bytes required to store the data. The expansion factor for a database character set indicates the maximum possible expansion in converting from UTF-8 to the character set. If the database character set is either UTF8 or AL32UTF8, then the expansion factor \( (\text{exp\_factor}) \) is 1. Otherwise, the expansion factor is equal to the maximum character size (measured in bytes) in the database character set.

Table 7–7 shows the database size limitations for SQL \textit{CHAR} data and the JDBC thin driver size restriction formulas for SQL \textit{CHAR} binds. Database limits are in bytes. Formulas determine the maximum allowed size of the UTF-8 encoding in bytes.

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Maximum Bind Size Allowed by Database</th>
<th>Formula for Determining the Maximum Bind Size, Measured in UTF-8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>2000 bytes</td>
<td>( 4000 / \text{exp_factor} )</td>
</tr>
<tr>
<td>VARCHAR2</td>
<td>4000 bytes</td>
<td>( 4000 / \text{exp_factor} )</td>
</tr>
<tr>
<td>LONG</td>
<td>( 2^{31} - 1 ) bytes</td>
<td>( (2^{31} - 1) / \text{exp_factor} )</td>
</tr>
</tbody>
</table>

The formulas guarantee that after the data is converted from UTF-8 to the database character set, the size of the data does not exceed the maximum size allowed in the database.

The number of UTF-16 characters that can be supported is determined by the number of bytes for each character in the data. All ASCII characters are one byte long in UTF-8 encoding. Other character types can be two or three bytes long.

Table 7–8 lists the expansion factors of some common server character sets. It also shows the JDBC thin driver maximum bind sizes for \textit{CHAR} and \textit{VARCHAR2} data for each character set.

<table>
<thead>
<tr>
<th>Server Character Set</th>
<th>Expansion Factor</th>
<th>JDBC Thin Driver Maximum Bind Size for SQL \textit{CHAR} Data, Measured in UTF-8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>WE8DEC</td>
<td>1</td>
<td>4000 bytes</td>
</tr>
</tbody>
</table>
Character Integrity Issues in a Multibyte Database Environment

Oracle JDBC drivers perform character set conversions as appropriate when character data is inserted into or retrieved from the database. The drivers convert Unicode characters used by Java clients to Oracle database character set characters, and vice versa. Character data that makes a round trip from the Java Unicode character set to the database character set and back to Java can suffer some loss of information. This happens when multiple Unicode characters are mapped to a single character in the database character set. An example is the Unicode full-width tilde character (0xFF5E) and its mapping to Oracle’s JA16SJIS character set. The round-trip conversion for this Unicode character results in the Unicode character 0x301C, which is a wave dash (a character commonly used in Japan to indicate range), not a tilde.

Figure 7–2 shows the round-trip conversion of the tilde character.

Figure 7–2  Character Integrity

<table>
<thead>
<tr>
<th>Server Character Set</th>
<th>Expansion Factor</th>
<th>JDBC Thin Driver Maximum Bind Size for SQL CHAR Data, Measured in UTF-8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA16SJIS</td>
<td>2</td>
<td>2000 bytes</td>
</tr>
<tr>
<td>JA16EUC</td>
<td>3</td>
<td>1333 bytes</td>
</tr>
<tr>
<td>AL32UTF8</td>
<td>1</td>
<td>4000 bytes</td>
</tr>
</tbody>
</table>
This issue is not a bug in Oracle's JDBC. It is an unfortunate side effect of the ambiguity in character mapping specification on different operating systems. Fortunately, this problem affects only a small number of characters in a small number of Oracle character sets such as JA16SJIS, JA16EUC, ZHT16BIG5, and KO16KS5601. The workaround is to avoid making a full round-trip with these characters.

**ODBC and OLE DB Programming with Unicode**

You should use the Oracle ODBC driver or Oracle Provider for OLE DB to access the Oracle server when using a Windows platform. This section describes how these drivers support Unicode. It includes the following topics:

- Unicode-Enabled Drivers in ODBC and OLE DB
- OCI Dependency in Unicode
- ODBC and OLE DB Code Conversion in Unicode
- ODBC Unicode Datatypes
- OLE DB Unicode Datatypes
- ADO Access

**Unicode-Enabled Drivers in ODBC and OLE DB**

Oracle’s ODBC driver and Oracle Provider for OLE DB can handle Unicode data properly without data loss. For example, you can run a Unicode ODBC application containing Japanese data on English Windows if you install Japanese fonts and an input method editor for entering Japanese characters.

Oracle provides ODBC and OLE DB products for Windows platforms only. For Unix platforms, contact your vendor.

**OCI Dependency in Unicode**

OCI Unicode binding and defining features are used by the ODBC and OLE DB drivers to handle Unicode data. OCI Unicode data binding and defining features are independent from NLS_LANG. This means Unicode data is handled properly, irrespective of the NLS_LANG setting on the platform.

See Also: "OCI Programming with Unicode" on page 7-13
**ODBC and OLE DB Code Conversion in Unicode**

In general, no redundant data conversion occurs unless you specify a different client datatype from that of the server. If you bind Unicode buffer `SQL_C_WCHAR` with a Unicode data column like `NCHAR`, for example, then ODBC and OLE DB drivers bypass it between the application and OCI layer.

If you do not specify datatypes before fetching, but call `SQLGetData` with the client datatypes instead, then the conversions in Table 7–9 occur.

**Table 7–9  ODBC Implicit Binding Code Conversions**

<table>
<thead>
<tr>
<th>Datatypes of ODBC Client Buffer</th>
<th>Datatypes of the Target Column in the Database</th>
<th>Fetch Conversions</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SQL_C_WCHAR</code></td>
<td>CHAR, VARCHAR2, CLOB</td>
<td><a href="#">If the database character set is a subset of the NLS_LANG character set, then the conversions occur in the following order:</a></td>
<td>No unexpected data loss May degrade performance if database character set is a subset of the NLS_LANG character set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Database character set</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NLS_LANG</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UTF-16 in OCI</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UTF-16 in ODBC</td>
<td></td>
</tr>
<tr>
<td><code>SQL_C_CHAR</code></td>
<td>CHAR, VARCHAR2, CLOB</td>
<td>If database character set is a subset of NLS_LANG character set: Database character set to NLS_LANG in OCI</td>
<td>No unexpected data loss May degrade performance if database character set is not a subset of NLS_LANG character set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Database character set, UTF-16, to NLS_LANG character set in OCI and ODBC</td>
<td></td>
</tr>
</tbody>
</table>

You must specify the datatype for inserting and updating operations.

The datatype of the ODBC client buffer is given when you call `SQLGetData` but not immediately. Hence, `SQLFetch` does not have the information.
Because the ODBC driver guarantees data integrity, if you perform implicit bindings, then redundant conversion may result in performance degradation. Your choice is the trade-off between performance with explicit binding or usability with implicit binding.

**OLE DB Code Conversions**

Unlike ODBC, OLE DB only enables you to perform implicit bindings for inserting, updating, and fetching data. The conversion algorithm for determining the intermediate character set is the same as the implicit binding cases of ODBC.

### Table 7–10  OLE DB Implicit Bindings

<table>
<thead>
<tr>
<th>Datatypes of OLE_DB Client Buffer</th>
<th>Datatypes of the Target Column in the Database</th>
<th>In-Binding and Out-Binding Conversions</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBTYPE_WCHAR</td>
<td>CHAR, VARCHAR2, CLOB</td>
<td>If database character set is a subset of the NLS_LANG character set: Database character set to and from NLS_LANG character set in OCI. NLS_LANG character set to UTF-16 in OLE DB</td>
<td>No unexpected data loss. May degrade performance if database character set is a subset of NLS_LANG character set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If database character set is NOT a subset of NLS_LANG character set: Database character set to and from UTF-16 in OCI</td>
<td></td>
</tr>
<tr>
<td>DBTYPE_CHAR</td>
<td>CHAR, VARCHAR2, CLOB</td>
<td>If database character set is a subset of the NLS_LANG character set: Database character set to and from NLS_LANG in OCI</td>
<td>No unexpected data loss. May degrade performance if database character set is not a subset of NLS_LANG character set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If database character set is not a subset of NLS_LANG character set: Database character set to and from UTF-16 in OCI. UTF-16 to NLS_LANG character set in OLE DB</td>
<td></td>
</tr>
</tbody>
</table>
ODBC Unicode Datatypes

In ODBC Unicode applications, use SQLWCHAR to store Unicode data. All standard Windows Unicode functions can be used for SQLWCHAR data manipulations. For example, wcslen counts the number of characters of SQLWCHAR data:

```c
SQLWCHAR sqlStmt[] = L"select ename from emp";
len = wcslen(sqlStmt);
```

Microsoft’s ODBC 3.5 specification defines three Unicode datatype identifiers for the SQL_C_WCHAR, SQL_C_WVARCHAR, and SQL_WLONGVARCHAR clients; and three Unicode datatype identifiers for servers SQL_WCHAR, SQL_WVARCHAR, and SQL_WLONGVARCHAR.

For binding operations, specify datatypes for both client and server using SQLBindParameter. The following is an example of Unicode binding, where the client buffer Name indicates that Unicode data (SQL_C_WCHAR) is bound to the first bind variable associated with the Unicode column (SQL_WCHAR):

```c
SQLBindParameter(StatementHandle, 1, SQL_PARAM_INPUT, SQL_C_WCHAR, SQL_WCHAR, NameLen, 0, (SQLPOINTER)Name, 0, &Name);
```

Table 7–11 represents the datatype mappings of the ODBC Unicode datatypes for the server against SQL NCHAR datatypes.

<table>
<thead>
<tr>
<th>ODBC Datatype</th>
<th>Oracle Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_WCHAR</td>
<td>NCHAR</td>
</tr>
<tr>
<td>SQL_WVARCHAR</td>
<td>NVARCHAR2</td>
</tr>
<tr>
<td>SQL_WLONGVARCHAR</td>
<td>NCLOB</td>
</tr>
</tbody>
</table>

According to ODBC specifications, SQL_WCHAR, SQL_WVARCHAR, and SQL_WLONGVARCHAR are treated as Unicode data, and are therefore measured in the number of characters instead of the number of bytes.

OLE DB Unicode Datatypes

OLE DB offers the wchar_t, BSTR, and OLESTR datatypes for a Unicode C client. In practice, wchar_t is the most common datatype and the others are for specific purposes. The following example assigns a static SQL statement:

```c
wchar_t *sqlStmt = OLESTR("SELECT ename FROM emp");
```
The `OLESTR` macro works exactly like an "L" modifier to indicate the Unicode string. If you need to allocate Unicode data buffer dynamically using `OLESTR`, then use the `IMalloc` allocator (for example, `CoTaskMemAlloc`). However, using `OLESTR` is not the normal method for variable length data; use `wchar_t*` instead for generic string types. `BSTR` is similar. It is a string with a length prefix in the memory location preceding the string. Some functions and methods can accept only `BSTR` Unicode datatypes. Therefore, `BSTR` Unicode string must be manipulated with special functions like `SysAllocString` for allocation and `SysFreeString` for freeing memory.

Unlike ODBC, OLE DB does not allow you to specify the server datatype explicitly. When you set the client datatype, the OLE DB driver automatically performs data conversion if necessary.

Table 7–12 illustrates OLE DB datatype mapping.

<table>
<thead>
<tr>
<th>OLE DB Datatype</th>
<th>Oracle Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DBTYPE_WCHAR</code></td>
<td><code>NCHAR</code> or <code>NVARCHAR2</code></td>
</tr>
</tbody>
</table>

If `DBTYPE_BSTR` is specified, then it is assumed to be `DBTYPE_WCHAR` because both are Unicode strings.

**ADO Access**

ADO is a high-level API to access database with the OLE DB and ODBC drivers. Most database application developers use the ADO interface on Windows because it is easily accessible from Visual Basic, the primary scripting language for Active Server Pages (ASP) for the Internet Information Server (IIS). To OLE DB and ODBC drivers, ADO is simply an OLE DB consumer or ODBC application. ADO assumes that OLE DB and ODBC drivers are Unicode-aware components; hence, it always attempts to manipulate Unicode data.

**XML Programming with Unicode**

XML support of Unicode is essential for software development for global markets so that text information can be exchanged in any language. Unicode uniformly supports almost every character and language, which makes it much easier to support multiple languages within XML. To enable Unicode for XML within an Oracle database, the character set of the database must be UTF-8. By enabling
Unicode text handling in your application, you acquire a basis for supporting any language. Every XML document is Unicode text and potentially multilingual, unless it is guaranteed that only a known subset of Unicode characters will appear on your documents. Thus Oracle recommends that you enable Unicode for XML. Unicode support comes with Java and many other modern programming environments.

This section includes the following topics:

- Writing an XML File in Unicode with Java
- Reading an XML File in Unicode with Java
- Parsing an XML Stream in Unicode with Java

## Writing an XML File in Unicode with Java

A common mistake in reading and writing XML files is using the `Reader` and `Writer` classes for character input and output. Using `Reader` and `Writer` for XML files should be avoided because it requires character set conversion based on the default character encoding of the runtime environment.

For example, using `FileWriter` class is not safe because it converts the document to the default character encoding. The output file can suffer from a parsing error or data loss if the document contains characters that are not available in the default character encoding.

UTF-8 is popular for XML documents, but UTF-8 is not usually the default file encoding for Java. Thus using a Java class that assumes the default file encoding can cause problems.

The following example shows how to avoid these problems:

```java
import java.io.*;
import oracle.xml.parser.v2.*;

public class I18nSafeXMLFileWritingSample
{
    public static void main(String[] args) throws Exception
    {
        // create a test document
        XMLDocument doc = new XMLDocument();
        doc.setVersion( "1.0" );
        doc.appendChild(doc.createComment( "This is a test empty document." ));
        doc.appendChild(doc.createElement( "root" ));
    }
}
```
// create a file
File file = new File( "myfile.xml" );

// create a binary output stream to write to the file just created
FileOutputStream fos = new FileOutputStream( file );

// create a Writer that converts Java character stream to UTF-8 stream
OutputStreamWriter osw = new OutputStreamWriter( fos, "UTF8" );

// buffering for efficiency
Writer w = new BufferedWriter( osw );

// create a PrintWriter to adapt to the printing method
PrintWriter out = new PrintWriter( w );

// print the document to the file through the connected objects
doc.print( out );
}

**Reading an XML File in Unicode with Java**

Do not read XML files as text input. When reading an XML document stored in a file system, use the parser to automatically detect the character encoding of the document. Avoid using a `Reader` class or specifying a character encoding on the input stream. Given a binary input stream with no external encoding information, the parser automatically figures out the character encoding based on the byte order mark and encoding declaration of the XML document. Any well-formed document in any supported encoding can be successfully parsed using the following sample code:

```java
import java.io.*;
import oracle.xml.parser.v2.*;

public class I18nSafeXMLFileReadingSample {
    public static void main(String[] args) throws Exception {
        // create an instance of the xml file
        File file = new File( "myfile.xml" );

        // create a binary input stream
        FileInputStream fis = new FileInputStream( file );
    }
}
```
// buffering for efficiency
BufferedInputStream in = new BufferedInputStream(fis);

// get an instance of the parser
DOMParser parser = new DOMParser();

// parse the xml file
parser.parse(in);

Parsing an XML Stream in Unicode with Java

When the source of an XML document is not a file system, the encoding information is usually available before reading the document. For example, if the input document is provided in the form of a Java character stream or Reader, its encoding is evident and no detection should take place. The parser can begin parsing a Reader in Unicode without regard to the character encoding.

The following is an example of parsing a document with external encoding information:

import java.io.*;
import java.net.*;
import org.xml.sax.*;
import oracle.xml.parser.v2.*;

public class I18nSafeXMLStreamReadingSample
{
    public static void main(String[] args) throws Exception
    {
        // create an instance of the xml file
        URL url = new URL("http://myhost/mydocument.xml");

        // create a connection to the xml document
        URLConnection conn = url.openConnection();

        // get an input stream
        InputStream is = conn.getInputStream();

        // buffering for efficiency
        BufferedInputStream bis = new BufferedInputStream(is);

        /* figure out the character encoding here */
        /* a typical source of encoding information is the content-type header */
/* we assume it is found to be utf-8 in this example */
String charset = "utf-8";

// create an InputSource for UTF-8 stream
InputSource in = new InputSource( bis );
in.setEncoding( charset );

// get an instance of the parser
DOMParser parser = new DOMParser();

// parse the xml stream
parser.parse( in );
}
This chapter includes the following sections:

- Overview of the Oracle Globalization Development Kit
- Designing a Global Internet Application
- Developing a Global Internet Application
- Getting Started with the Globalization Development Kit
- GDK Application Framework for J2EE
- GDK Java API
- The GDK Application Configuration File
- GDK for Java Supplied Packages and Classes
- GDK for PL/SQL Supplied Packages
- GDK Error Messages
Overview of the Oracle Globalization Development Kit

Designing and developing a globalized application can be a daunting task even for the most experienced developers. This is usually caused by lack of knowledge and the complexity of globalization concepts and APIs. Application developers who write applications using the Oracle database need to understand the Globalization Support architecture of the database, including the properties of the different character sets, territories, languages and linguistic sort definitions. They also need to understand the globalization functionality of their middle-tier programming environment, and find out how it can interact and synchronize with the locale model of the database. Finally, to develop a globalized Internet application, they need to design and write code that is capable of simultaneously supporting multiple clients running on different operating systems with different character sets and locale requirements.

Oracle Globalization Development Kit (GDK) simplifies the development process and reduces the cost of developing Internet applications that will be used to support a global environment.

This release of the GDK includes comprehensive programming APIs for both Java and PL/SQL, code samples, and documentation that address many of the design, development, and deployment issues encountered while creating global applications.

The GDK mainly consists of two parts: GDK for Java and GDK for PL/SQL. GDK for Java provides globalization support to Java applications. GDK for PL/SQL provides globalization support to the PL/SQL programming environment. The features offered in GDK for Java and GDK for PL/SQL are not identical.

Designing a Global Internet Application

There are two architectural models for deploying a global Web site or a global Internet application, depending on your globalization and business requirements. Which model to deploy affects how the Internet application is developed and how the application server is configured in the middle-tier. The two models are:

- Multiple instances of monolingual Internet applications

  Internet applications that support only one locale in a single binary are classified as monolingual applications. A locale refers to a national language and the region in which the language is spoken. For example, the primary language of the United States and Great Britain is English. However, the two territories have different currencies and different conventions for date formats.
Therefore, the United States and Great Britain are considered to be two different locales.

This level of globalization support is suitable for customers who want to support one locale for each instance of the application. Users need to have different entry points to access the applications for different locales. This model is manageable only if the number of supported locales is small.

- Single instance of a multilingual application

Internet applications that support multiple locales simultaneously in a single binary are classified as multilingual applications. This level of globalization support is suitable for customers who want to support several locales in an Internet application simultaneously. Users of different locale preferences use the same entry point to access the application.

Developing an application using the monolingual model is very different from developing an application using the multilingual model. The Globalization Development Kit consists of libraries, which can assist in the development of global applications using either architectural model.

The rest of this section includes the following topics:

- Deploying a Monolingual Internet Application
- Deploying a Multilingual Internet Application

**Deploying a Monolingual Internet Application**

Deploying a global Internet application with multiple instances of monolingual Internet applications is shown in Figure 8–1.
Each application server is configured for the locale that it serves. This deployment model assumes that one instance of an Internet application runs in the same locale as the application in the middle tier.

The Internet applications access a back-end database in the native encoding used for the locale. The following are advantages of deploying monolingual Internet applications:

- The support of the individual locales is separated into different servers so that multiple locales can be supported independently in different locations and that the workload can be distributed accordingly. For example, customers may want...
to support Western European locales first and then support Asian locales such as Japanese (Japan) later.

- The complexity required to support multiple locales simultaneously is avoided. The amount of code to write is significantly less for a monolingual Internet application than for a multilingual Internet application.

The following are disadvantages of deploying monolingual Internet applications:

- Extra effort is required to maintain and manage multiple servers for different locales. Different configurations are required for different application servers.

- The minimum number of application servers required depends on the number of locales the application supports, regardless of whether the site traffic will reach the capacity provided by the application servers.

- Load balancing for application servers is limited to the group of application servers for the same locale.

- More QA resources, both human and machine, are required for multiple configurations of application servers. Internet applications running on different locales must be certified on the corresponding application server configuration.

- It is not designed to support multilingual content. For example, a web page containing Japanese and Arabic data cannot be easily supported in this model.

As more and more locales are supported, the disadvantages quickly outweigh the advantages. With the limitation and the maintenance overhead of the monolingual deployment model, this deployment architecture is suitable for applications that support only one or two locales.

**Deploying a Multilingual Internet Application**

Multilingual Internet applications are deployed to the application servers with a single application server configuration that works for all locales. Figure 8–2 shows the architecture of a multilingual Internet application.
To support multiple locales in a single application instance, the application may need to do the following:

- Dynamically detect the locale of the users and adapt to the locale by constructing HTML pages in the language and cultural conventions of the locale
- Process character data in Unicode so that data in any language can be supported. Character data can be entered by users or retrieved from back-end databases.
Dynamically determine the HTML page encoding (or character set) to be used for HTML pages and convert content from Unicode to the page encoding and the reverse.

The following are major advantages of deploying multilingual Internet application:

- Using a single application server configuration for all application servers simplifies the deployment configuration and hence reduces the cost of maintenance.
- Performance tuning and capacity planning do not depend on the number of locales supported by the Web site.
- Introducing additional locales is relatively easy. No extra machines are necessary for the new locales.
- Testing the application across different locales can be done in a single testing environment.
- This model can support multilingual content within the same instance of the application. For example, a web page containing Japanese, Chinese, English and Arabic data can be easily supported in this model.

The disadvantage of deploying multilingual Internet applications is that it requires extra coding during application development to handle dynamic locale detection and Unicode, which is costly when only one or two languages need to be supported.

Deploying multilingual Internet applications is more appropriate than deploying monolingual applications when Web sites support multiple locales.

Developing a Global Internet Application

Building an Internet application that supports different locales requires good development practices.

For multilingual Internet applications, 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. The application server then communicates with the database server, exchanging data while maintaining the preferences of the different locales and character set settings. One of the main considerations when developing a multilingual internet application is to be able to dynamically detect, cache, and provide the appropriate contents according to the user’s preferred locale.
For monolingual Internet applications, the locale of the user is always fixed and usually follows the default locale of the runtime environment. Hence the locale configuration is much simpler.

The following sections describe some of the most common issues that developers encounter when building a global Internet application:

- Locale Determination
- Locale Awareness
- Localizing the Content

**Locale Determination**

To be locale-aware or locale-sensitive, Internet applications need to be able to determine the preferred locale of the user.

Monolingual applications always serve users with the same locale, and that locale should be equivalent to the default runtime locale of the corresponding programming environment.

Multilingual applications can determine a user locale dynamically in three ways. Each method has advantages and disadvantages, but they can be used together in the applications to complement each other. The user locale can be determined in the following ways:

- Based on the user profile information from a LDAP directory server such as the Oracle Internet Directory or other user profile tables stored inside the database
  
  The schema for the user profile should include preferred locale attribute to indicate the locale of a user. This way of determining a locale user does not work if a user has not been logged on before.

- Based on the default locale of the browser
  
  Get the default ISO locale setting from a browser. The default ISO locale of the browser is sent through the Accept-Language HTTP header in every HTTP request. If the Accept-Language header is NULL, then the desired locale should default to English. The drawback of this approach is that the Accept-Language header may not be a reliable source of information for the locale of a user.

- Based on user selection
  
  Allow users to select a locale from a list box or from a menu, and switch the application locale to the one selected.
The Globalization Development Kit provides an application framework that enables you to use these locale determination methods declaratively.

**See Also:** "Getting Started with the Globalization Development Kit" on page 8-10

### Locale Awareness

To be locale-aware or locale-sensitive, Internet applications need to determine the locale of a user. After the locale of a user is determined, applications should:

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

Locale-sensitive functions, such as date, time, and monetary formatting, are built into various programming environments such as Java and PL/SQL. Applications may use them to format the HTML pages according to the cultural conventions of the locale of a user. A locale is represented differently in different programming environments. For example, the French (Canada) locale is represented in different environments as follows:

- In the ISO standard, it is represented by \texttt{fr-CA} where \texttt{fr} is the language code defined in the ISO 639 standard and \texttt{CA} is the country code defined in the ISO 3166 standard.

- In Java, it is represented as a Java locale object constructed with \texttt{fr}, the ISO language code for French, as the language and \texttt{CA}, the ISO country code for Canada, as the country. The Java locale name is \texttt{fr_CA}.

- In PL/SQL and SQL, it is represented mainly by the \texttt{NLS_LANGUAGE} and \texttt{NLS_TERRITORY} session parameters where the value of the \texttt{NLS_LANGUAGE} parameter is equal to \texttt{CANADIAN FRENCH} and the value of the \texttt{NLS_TERRITORY} parameter is equal to \texttt{CANADA}.

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 \texttt{NLS_LANGUAGE} and \texttt{NLS_TERRITORY} values and change the parameter values to match the user’s locale before calling the PL/SQL procedures.

The Globalization Development Kit for Java provides a set of Java classes to ensure consistency on locale-sensitive behaviors with Oracle databases.
Localizing the Content

For the application to support a multilingual environment, it must be able to present the content in the preferred language and in the locale convention of the user. Hard-coded user interface text must first be externalized from the application, together with any image files, so that they can be translated into the different languages supported by the application. The translation files then must be staged in separate directories, and the application must be able to locate the relevant content according to the user locale setting. Special application handling may also be required to support a fallback mechanism, so that if the user-preferred locale is not available, then the next most suitable content is presented. For example, if Canadian French content is not available, then it may be suitable for the application to switch to the French files instead.

Getting Started with the Globalization Development Kit

The Globalization Development Kit (GDK) for Java provides a J2EE application framework and Java APIs to develop globalized Internet applications using the best globalization practices and features designed by Oracle. It reduces the complexities and simplifies the code that Oracle developers require to develop globalized Java applications.

GDK for Java complements the existing globalization features in J2EE. Although the J2EE platform already provides a strong foundation for building globalized applications, its globalization functionalities and behaviors can be quite different from Oracle’s functionalities. GDK for Java provides synchronization of locale-sensitive behaviors between the middle-tier Java application and the database server.

GDK for PL/SQL contains a suite of PL/SQL packages that provide additional globalization functionalities for applications written in PL/SQL.

Figure 8–3 shows the major components of the GDK and how they are related to each other. User applications run on the J2EE container of Oracle Application Server in the middle tier. GDK provides the application framework that the J2EE application uses to simplify coding to support globalization. Both the framework and the application call the GDK Java API to perform locale-sensitive tasks. GDK for PL/SQL offers PL/SQL packages that help to resolve globalization issues specific to the PL/SQL environment.
The functionalities offered by GDK for Java can be divided into two categories:

- The GDK application framework for J2EE provides the globalization framework for building J2EE-based Internet application. The framework encapsulates the complexity of globalization programming, such as determining user locale, maintaining locale persistency, and processing locale information. It consists of a set of Java classes through which applications can gain access to the framework. These associated Java classes enable applications to code against the framework so that globalization behaviors can be extended declaratively.

- The GDK Java API offers development support in Java applications and provides consistent globalization operations as provided in Oracle database servers. The API is accessible and is independent of the GDK framework so that
standalone Java applications and J2EE applications that are not based on the GDK framework are able to access the individual features offered by the Java API. The features provided in the Java API include data and number formatting, sorting, and handling character sets in the same way as the Oracle Database.

**Note:** The GDK Java API is certified with JDK versions 1.3 and later with the following exception: The character set conversion classes depend on the java.nio.charset package, which is available in JDK 1.4 and later.

GDK for Java is contained in two files: orai18n.jar and orai18n-lcsd.jar. The files are shipped with the Oracle database. If the application using the GDK is not hosted on the same machine as the database, then the GDK files must be copied to the application server and included into the CLASSPATH to run your application. You do not need to install the Oracle Database into your application server to be able to run the GDK inside your Java application. GDK is a pure Java library that runs on every platform. The Oracle client parameters NLS_LANG and ORACLE_HOME are not required.

**GDK Application Framework for J2EE**

GDK for Java provides the globalization framework for middle-tier J2EE applications. The framework encapsulates the complexity of globalization programming, such as determining user locale, maintaining locale persistency, and processing locale information. This framework minimizes the effort required to make Internet applications global-ready. The GDK application framework is shown in Figure 8–4.
The main Java classes composing the framework are as follows:

- `ApplicationContext` provides the globalization context of an application. The context information includes the list of supported locales and the rule for determining user-preferred locale. The context information is obtained from the GDK application configuration file for the application.

- The set of `LocaleSource` classes can be plugged into the framework. Each `LocaleSource` class implements the `LocaleSource` interface to get the locale from the corresponding source. Oracle bundles several `LocaleSource` classes in GDK. For example, the `DBLocaleSource` class obtains the locale information of the current user from a database schema. You can also write a customized `LocaleSource` class by implementing the same `LocaleSource` interface and plugging it into the framework.

- `ServletRequestWrapper` and `ServletResponseWrapper` are the main classes of the GDK Servlet filter that transforms HTTP requests and HTTP responses. `ServletRequestWrapper` instantiates a `Localizer` object for each HTTP request based on the information gathered from the `ApplicationContext` and `LocaleSource` objects and ensures that forms
parameters are handled properly. `ServletResponseWrapper` controls how HTTP response should be constructed.

- **Localizer** is the all-in-one object that exposes the important functions that are sensitive to the current user locale and application context. It provides a centralized set of methods for you to call and make your applications behave appropriately to the current user locale and application context.

- The GDK Java API is always available for applications to enable finer control of globalization behavior.

The GDK application framework simplifies the coding required for your applications to support different locales. When you write a J2EE application according to the application framework, the application code is independent of what locales the application supports, and you control the globalization support in the application by defining it in the GDK application configuration file. There is no code change required when you add or remove a locale from the list of supported application locales.

The following list gives you some idea of the extent to which you can define the globalization support in the GDK application configuration file:

- You can add and remove a locale from the list of supported locales.
- You can change the way the user locale is determined.
- You can change the HTML page encoding of your application.
- You can specify how the translated resources can be located.
- You can plug a new `LocaleSource` object into the framework and use it to detect a user locale.

This section includes the following topics:

- Making the GDK Framework Available to J2EE Applications
- Integrating Locale Sources into the GDK Framework
- Getting the User Locale From the GDK Framework
- Implementing Locale Awareness Using the GDK Localizer
- Defining the Supported Application Locales in the GDK
- Handling Non-ASCII Input and Output in the GDK Framework
- Managing Localized Content in the GDK
Making the GDK Framework Available to J2EE Applications

The behavior of the GDK application framework for J2EE is controlled by the GDK application configuration file, `gdkapp.xml`. The application configuration file allows developers to specify the behaviors of globalized applications in one centralized place. One application configuration file is required for each J2EE application using the GDK. The `gdkapp.xml` file should be placed in the `./WEB-INF` directory of the J2EE environment of the application. The file dictates the behavior and the properties of the GDK framework and the application that is using it. It contains locale mapping tables, character sets of content files, and globalization parameters for the configuration of the application. The application administrator can modify the application configuration file to change the globalization behavior in the application, without needing to change the programs and to recompile them.

See Also: "The GDK Application Configuration File" on page 8-36

For a J2EE application to use the GDK application framework defined by the corresponding GDK application configuration file, the GDK Servlet filter and the GDK context listener must be defined in the `web.xml` file of the application. The `web.xml` file should be modified to include the following at the beginning of the file:

```xml
<web-app>
 <!-- Add GDK filter that is called after the authentication -->

<filter>
   <filter-name>gdkfilter</filter-name>
   <filter-class>oracle.i18n.servlet.filter.ServletFilter</filter-class>
</filter>

<filter-mapping>
   <filter-name>gdkfilter</filter-name>
   <url-pattern>*.jsp</url-pattern>
</filter-mapping>

<!-- Include the GDK context listener -->

<listener>
   <listener-class>oracle.i18n.servlet.listener.ContextListener</listener-class>
</listener>

</web-app>
```

Examples of the `gdkapp.xml` and `web.xml` files can be found in the `$ORACLE_HOME/nls/gdk/demo` directory.

See Also: "The GDK Application Configuration File" on page 8-36
The GDK application framework supports Servlet container version 2.3 and later. It uses the Servlet filter facility for transparent globalization operations such as determining the user locale and specifying the character set for content files. The ApplicationContextListener instantiates GDK application parameters described in the GDK application configuration file. The ServletFilter overrides the request and response objects with a GDK request (ServletRequestWrapper) and response (ServletResponseWrapper) objects, respectively.

If other application filters used in the application also override the same methods, then the filter in the GDK framework may return incorrect results. For example, if getLocale returns en_US, but the result is overridden by other filters, then the result of the GDK locale detection mechanism is affected. All of the methods that are being overridden in the filter of the GDK framework are documented in Oracle Globalization Development Kit Java API Reference. Be aware of potential conflicts when using other filters together with the GDK framework.

### Integrating Locale Sources into the GDK Framework

Determining the user’s preferred locale is the first step in making an application global-ready. The locale detection offered by the J2EE application framework is primitive. It lacks the method that transparently retrieves the most appropriate user locale among locale sources. It provides locale detection by the HTTP language preference only, and it cannot support a multilevel locale fallback mechanism. The GDK application framework provides support for predefined locale sources to complement J2EE. In a web application, several locale sources are available. Table 8–1 summarizes locale sources that are provided by the GDK.

<table>
<thead>
<tr>
<th>Locale</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP language preference</td>
<td>Locales included in the HTTP protocol as a value of Accept-Language. This is set at the web browser level. A locale fallback operation is required if the browser locale is not supported by the application.</td>
</tr>
<tr>
<td>User input locale</td>
<td>Locale specified by the user from a menu or a parameter in the HTTP protocol</td>
</tr>
<tr>
<td>User profile locale preference from database</td>
<td>Locale preference stored in the database as part of the user profiles</td>
</tr>
<tr>
<td>Application default locale</td>
<td>A locale defined in the GDK application configuration file. This locale is defined as the default locale for the application. Typically, this is used as a fallback locale when the other locale sources are not available.</td>
</tr>
</tbody>
</table>
GDK Application Framework for J2EE

See Also: "The GDK Application Configuration File" on page 8-36 for information about the GDK multilevel locale fallback mechanism

The GDK application framework provides seamless support for predefined locale sources, such as user input locale, HTTP language preference, user profile locale preference in the database, and the application default locale. You can incorporate the locale sources to the framework by defining them under the <locale-determine-rule> tag in the GDK application configuration file as follows:

```xml
<locale-determine-rule>
  <locale-source>oracle.i18n.servlet.localesource.UserInput</locale-source>
  <locale-source>oracle.i18n.servlet.localesource.HTTPAcceptLanguage</locale-source>
</locale-determine-rule>
```

Custom locale sources, such as locale preference from an LDAP server, can be easily implemented and integrated into the GDK framework. You need to implement the LocaleSource interface and specify the corresponding implementation class under the <locale-determine-rule> tag in the same way as the predefined locale sources were specified.

The LocaleSource implementation not only retrieves the locale information from the corresponding source to the framework but also updates the locale information to the corresponding source when the framework tells it to do so. Locale sources can be read-only or read/write, and they can be cacheable or noncacheable. The GDK framework initiates updates only to read/write locale sources and caches the locale information from cacheable locale sources. Examples of custom locale sources can be found in the $ORACLE_HOME/nls/gdk/demo directory.

See Also: Oracle Globalization Development Kit Java API Reference for more information about implementing a LocaleSource

Getting the User Locale From the GDK Framework

The GDK offers automatic locale detection to determine the current locale of the user. For example, the following code retrieves the current user locale in Java. It uses a Locale object explicitly.

```java
Locale loc = request.getLocale();
```

The getLocale() method returns the Locale that represents the current locale. This is similar to invoking the HttpServletRequest.getLocale() method in JSP or Java Servlet code. However, the logic in determining the user locale is
different, because multiple locale sources are being considered in the GDK framework.

Alternatively, you can get a Localizer object that encapsulates the Locale object determined by the GDK framework. For the benefits of using the Localizer object, see "Implementing Locale Awareness Using the GDK Localizer" on page 8-19.

```java
Localizer localizer = ServletHelper.getLocalizerInstance(request);
Locale loc = localizer.getLocale();
```

The locale detection logic of the GDK framework depends on the locale sources defined in the GDK application configuration file. The names of the locale sources are registered in the application configuration file. The following example shows the locale determination rule section of the application configuration file. It indicates that the user-preferred locale can be determined from either the LDAP server or from the HTTP Accept-Language header. The LDAPUserSchema locale source class should be provided by the application. Note that all of the locale source classes have to be extended from the LocaleSource abstract class.

```xml
<locale-determine-rule>
    <locale-source>LDAPUserSchema</locale-source>
    <locale-source>oracle.i18n.localesource.HTTPAcceptLanguage</locale-source>
</locale-determine-rule>
```

For example, when the user is authenticated in the application and the user locale preference is stored in an LDAP server, then the LDAPUserSchema class connects to the LDAP server to retrieve the user locale preference. When the user is anonymous, then the HttpAcceptLanguage class returns the language preference of the web browser.

The cache is maintained for the duration of a HTTP session. If the locale source is obtained from the HTTP language preference, then the locale information is passed to the application in the HTTP Accept-Language header and not cached. This enables flexibility so that the locale preference can change between requests. The cache is available in the HTTP session.

The GDK framework exposes a method for the application to overwrite the locale preference information persistently stored in locale sources such as the LDAP server or the user profile table in the database. This method also resets the current locale information stored inside the cache for the current HTTP session. The following is an example of overwriting the preferred locale using the `store` command.

```html
<input type="hidden"
name="<%=appctx.getParameterName(LocaleSource.Parameter.COMMAND)%>"/>
```
To discard the current locale information stored inside the cache, the `clean` command can be specified as the input parameter. The following table shows the list of commands supported by the GDK:

<table>
<thead>
<tr>
<th>Command</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>store</code></td>
<td>Updates user locale preferences in the available locale sources with the specified locale information. This command is ignored by the read-only locale sources.</td>
</tr>
<tr>
<td><code>clean</code></td>
<td>Discards the current locale information in the cache</td>
</tr>
</tbody>
</table>

Note that the GDK parameter names can be customized in the application configuration file to avoid name conflicts with other parameters used in the application.

**Implementing Locale Awareness Using the GDK Localizer**

The `Localizer` object obtained from the GDK application framework is an all-in-one globalization object that provides access to functions that are commonly used in building locale awareness in your applications. In addition, it provides functions to get information about the application context, such as the list of supported locales. The `Localizer` object simplifies and centralizes the code required to build consistent locale awareness behavior in your applications.

The `oracle.i18n.servlet` package contains the `Localizer` class. You can get the `Localizer` instance as follows:

```java
Localizer lc = ServletHelper.getLocalizerInstance(request);
```

The `Localizer` object encapsulates the most commonly used locale-sensitive information determined by the GDK framework and exposes it as locale-sensitive methods. This object includes the following functionalities pertaining to the user locale:

- Format date in long and short formats
- Format numbers and currencies
- Get collation key value of a string
- Get locale data such as language, country and currency names
- Get locale data to be used for constructing user interface
GDK Application Framework for J2EE

- Get a translated message from resource bundles
- Get text formatting information such as writing direction
- Encode and decode URLs
- Get the common list of time zones and linguistic sorts

For example, when you want to display a date in your application, you may want to call the `Localizer.formatDate()` or `Localizer.formatDateTime()` methods. When you want to determine the writing direction of the current locale, you can call the `Localizer.getWritingDirection()` and `Localizer.getAlignment()` to determine the value used in the `<DIR>` tag and `<ALIGN>` tag respectively.

The `Localizer` object also exposes methods to enumerate the list of supported locales and their corresponding languages and countries in your applications.

The `Localizer` object actually makes use of the classes in the GDK Java API to accomplish its tasks. These classes includes, but are not limited to, the following: `OraDateFormat`, `OraNumberFormat`, `OraCollator`, `OraLocaleInfo`, `oracle.i18n.util.LocaleMapper`, `oracle.i18n.net.URLEncoder`, and `oracle.i18n.net.URLDecoder`.

The `Localizer` object simplifies the code you need to write for locale awareness. It maintains caches of the corresponding objects created from the GDK Java API so that the calling application does not need to maintain these objects for subsequent calls to the same objects. If you require more than the functionality the `Localizer` object can provide, then you can always call the corresponding methods in the GDK Java API directly.

**See Also:** Oracle Globalization Development Kit Java API Reference for detailed information about the `Localizer` object

### Defining the Supported Application Locales in the GDK

The number of locales and the names of the locales that an application needs to support are based on the business requirements of the application. The names of the locales that are supported by the application are registered in the application configuration file. The following example shows the application locales section of the application configuration file. It indicates that the application supports German (de), Japanese (ja), and English for the US (en-US), with English defined as the default fallback application locale.

```xml
<application-locales>
  <locale>de</locale>
</application-locales>```
When the GDK framework detects the user locale, it verifies whether the locale that is returned is one of the supported locales in the application configuration file. The verification algorithm is as follows:

1. Retrieve the list of supported application locales from the application configuration file.
2. Check whether the locale that was detected is included in the list. If it is included in the list, then use this locale as the current client's locale.
3. If there is a variant in the locale that was detected, then remove the variant and check whether the resulting locale is in the list. For example, de_DE_EURO has a EURO variant. Remove the variant so that the resulting locale is de_DE.
4. If the locale includes a country code, then remove the country code and check whether the resulting locale is in the list. For example, de_DE has a country code of DE. Remove the country code so that the resulting locale is de.
5. If the detected locale does not match any of the locales in the list, then use the default locale that is defined in the application configuration file as the client locale.

By performing steps 3 and 4, the application can support users with the same language requirements but with different locale settings than those defined in the application configuration file. For example, the GDK can support de-AT (the Austrian variant of German), de-CH (the Swiss variant of German), and de-LU (the Luxembourgian variant of German) locales.

The locale fallback detection in the GDK framework is similar to that of the Java Resource Bundle, except that it is not affected by the default locale of the Java VM. This exception occurs because the Application Default Locale can be used during the GDK locale fallback operations.

If the application-locales section is omitted from the application configuration file, then the GDK assumes that the common locales, which can be returned from the OraLocaleInfo.getCommonLocales method, are supported by the application.

**Handling Non-ASCII Input and Output in the GDK Framework**

The character set (or character encoding) of an HTML page is a very important piece of information to a browser and an Internet application. The browser needs to
interpret this information so that it can use correct fonts and character set mapping tables for displaying pages. The Internet applications need to know so they can safely process input data from a HTML form based on the specified encoding.

The page encoding can be translated as the character set used for the locale to which an Internet application is serving. In order to correctly specify the page encoding for HTML pages without using the GDK framework, Internet applications must:

1. Determine the desired page input data character set encoding for a given locale.
2. Specify the corresponding encoding name for each HTTP request and HTTP response.

Applications using the GDK framework can ignore these steps. No application code change is required. The character set information is specified in the GDK application configuration file. At runtime, the GDK automatically sets the character sets for the request and response objects. The GDK framework does not support the scenario where the incoming character set is different from that of the outgoing character set.

The GDK application framework supports the following scenarios for setting the character sets of the HTML pages:

- A single local character set is dedicated to the whole application. This is appropriate for a monolingual Internet application. Depending on the properties of the character set, it may be able to support more than one language. For example, most Western European languages can be served by ISO-8859-1.

- Unicode UTF-8 is used for all contents regardless of the language. This is appropriate for a multilingual application that uses Unicode for deployment.

- The native character set for each language is used. For example, English contents are represented in ISO-8859-1, and Japanese contents are represented in Shift_JIS. This is appropriate for a multilingual Internet application that uses a default character set mapping for each locale. This is useful for applications that need to support different character sets based on the user locales. For example, for mobile applications that lack Unicode fonts or Internet browsers that cannot fully support Unicode, the character sets must be determined for each request.

The character set information is specified in the GDK application configuration file. The following is an example of setting UTF-8 as the character set for all the application pages.

```xml
<page-charset>UTF-8</page-charset>
```
The page character set information is used by the ServletRequestWrapper class, which sets the proper character set for the request object. It is also used by the ContentType HTTP header specified in the ServletResponseWrapper class for output when instantiated. If page-charset is set to AUTO-CHARSET, then the character set is assumed to be the default character set for the current user locale. Set page-charset to AUTO-CHARSET as follows:

```xml
<page-charset>AUTO-CHARSET</page-charset>
```

The default mappings are derived from the LocaleMapper class, which provides the default IANA character set for the locale name in the GDK Java API.

Table 8–2 lists the mappings between the common ISO locales and their IANA character sets.

<table>
<thead>
<tr>
<th>ISO Locale</th>
<th>NLS LANGUANGE Value</th>
<th>NLS TERRITORY Value</th>
<th>IANA Character Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>ar-SA</td>
<td>ARABIC</td>
<td>SAUDI ARABIA</td>
<td>WINDOWS-1256</td>
</tr>
<tr>
<td>de-DE</td>
<td>GERMAN</td>
<td>GERMANY</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>en-US</td>
<td>AMERICAN</td>
<td>AMERICA</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>en-GB</td>
<td>ENGLISH</td>
<td>UNITED KINGDOM</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>el</td>
<td>GREEK</td>
<td>GREECE</td>
<td>WINDOWS-1253</td>
</tr>
<tr>
<td>es-ES</td>
<td>SPANISH</td>
<td>SPAIN</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>fr</td>
<td>FRENCH</td>
<td>FRANCE</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>fr-CA</td>
<td>CANADIAN FRENCH</td>
<td>CANADA</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>iw</td>
<td>HEBREW</td>
<td>ISRAEL</td>
<td>WINDOWS-1255</td>
</tr>
<tr>
<td>ko</td>
<td>KOREAN</td>
<td>KOREA</td>
<td>EUC-KR</td>
</tr>
<tr>
<td>ja</td>
<td>JAPANESE</td>
<td>JAPAN</td>
<td>SHIFT_JIS</td>
</tr>
<tr>
<td>it</td>
<td>ITALIAN</td>
<td>ITALY</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>pt</td>
<td>PORTUGUESE</td>
<td>PORTUGAL</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>pt-BR</td>
<td>BRAZILIAN PORTUGUESE</td>
<td>BRAZIL</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>tr</td>
<td>TURKISH</td>
<td>TURKEY</td>
<td>WINDOWS-1254</td>
</tr>
<tr>
<td>nl</td>
<td>DUTCH</td>
<td>THE NETHERLANDS</td>
<td>WINDOWS-1252</td>
</tr>
</tbody>
</table>
The locale to character set mapping in the GDK is also be customized. To override the default mapping defined in the GDK Java API, a locale-to-character-set mapping table can be specified in the application configuration file.

```xml
<locale-charset-maps>
  <locale-charset>
    <locale>ja</locale><charset>EUC-JP</charset>
  </locale-charset>
</locale-charset-maps>
```

The previous example shows that for locale Japanese (ja), the GDK changes the default character set from SHIFT_JIS to EUC-JP.

**See Also:** "Oracle Locale Information in the GDK" on page 8-27

### Managing Localized Content in the GDK

This section includes the following topics:

- Managing Localized Content in JSPs and Java Servlets
- Managing Localized Content in Static Files

#### Managing Localized Content in JSPs and Java Servlets

Resource bundles enable access to localized contents at runtime in J2SE. Translatable strings within Java servlets and Java Server Pages (JSPs) are externalized into Java resource bundles so that these resource bundles can be translated independently into different languages. The translated resource bundles carry the same base class names as the English bundles, using the Java locale name as the suffix.

To retrieve translated data from the resource bundle, the `getBundle()` method must be invoked for every request.

```jsp
<% Locale user_locale=request.getLocale();
    ResourceBundle rb=ResourceBundle.getBundle("resource",user_locale); %>
<%= rb.getString("Welcome") %>
```
The GDK framework simplifies the retrieval of text strings from the resource bundles. `Localizer.getMessage()` is a wrapper to the resource bundle.

```jsp
<% Localizer.getMessage ("Welcome") %>
```

Instead of specifying the base class name as `getBundle()` in the application, you can specify the resource bundle in the application configuration file, so that the GDK automatically instantiates a `ResourceBundle` object when a translated text string is requested.

```xml
<message-bundles>
  <resource-bundle name="default">resource</resource-bundle>
</message-bundles>
```

Multiple resource bundles can be specified in the configuration file. To access a nondefault bundle, specify the `name` parameter in the `getMessage` method.

### Managing Localized Content in Static Files

For a application, which supports only one locale, the URL that has a suffix of `/index.html` typically takes the user to the starting page of the application.

In a globalized application, contents in different languages are usually stored separately, and it is common for them to be staged in different directories or with different file names based on the language or the country name. This information is then used to construct the URLs for localized content retrieval in the application.

The following examples illustrate how to retrieve the French and Japanese versions of the index page. Their suffixes are as follows:

- `/fr/index.html`
- `/ja/index.html`

By using the `rewriteURL()` method, the GDK framework handles the logic to locate the translated files from the corresponding language directories. The `ServletHelper.rewriteURL()` method rewrites a URL based on the rules specified in the application configuration file. This method is used to determine the correct location where the localized content is staged.

The following is an example of the JSP code:

```jsp
<img src="<%=ServletHelper.rewriteURL("image/welcome.jpg", request)%>">
<a href="<%=ServletHelper.rewriteURL("html/welcome.html", request)%>">
```

The URL rewrite definitions are defined in the GDK application configuration file:

```xml
<url-rewrite-rule fallback="yes">
```

```xml
</url-rewrite-rule>
```
The pattern section defined in the rewrite rule follow the regular expression conventions. The result section supports the following special variables for replacing:

- \$L is used to represent the ISO 639 language code part of the current user locale
- \$C represents the ISO 3166 country code
- \$A represents the entire locale string, where the ISO 639 language code and ISO 3166 country code are connected with an underscore character (_)
- \$1 to \$9 represent the matched substrings

For example, if the current user locale is ja, then the URL for the welcome.jpg image file is rewritten as image/ja/welcome.jpg, and welcome.html is changed to html/ja/welcome.html.

Both ServletHelper.rewriteURL() and Localizer.getMessage() methods perform consistent locale fallback operations in the case where the translation files for the user locale are not available. For example, if the online help files are not available for the es_MX locale (Spanish for Mexico), but the es (Spanish for Spain) files are available, then the methods will select the Spanish translated files as the substitute.

GDK Java API

Java's globalization functionalities and behaviors are not the same as those offered in the database. For example, J2SE supports a set of locales and character sets that are different from Oracle’s locales and character sets. This inconsistency can be confusing for users when their application contains data that is formatted based on 2 different conventions. For example, dates that are retrieved from the database are formatted using Oracle conventions, (such as number and date formatting and linguistic sort ordering), but the static application data are typically formatted using Java locale conventions. Java’s globalization functionalities can also be different depending on the version of the JDK that the application runs on.

Before Oracle Database 10g, when an application was required to incorporate Oracle globalization features, it had to make connections to the database server and issue SQL statements. Such operations make the application complicated and generate more network connections to the database server.
The GDK Java API extends Oracle’s database globalization features to the middle tier. By enabling applications to perform globalization logic such as Oracle date and number formatting and linguistic sorting in the middle tier, the GDK Java API allows developers to eliminate expensive programming logic in the database, hence improving the overall application performance by reducing the database load in the database server and the unnecessary network traffic between the application tier and the database server.

The GDK Java API also offers advance globalization functionalities, such as language and character set detection, and the enumeration of common locale data for a territory or a language (for example, all time zones supported in Canada). These are globalization features that are not available in most programming platforms. Without the GDK Java API, developers must write business logic to handle them inside an application.

The following are the key functionalities of the GDK Java API:

- Oracle Locale Information in the GDK
- Oracle Locale Mapping in the GDK
- Oracle Character Set Conversion (JDK 1.4 and Later) in the GDK
- Oracle Date, Number, and Monetary Formats in the GDK
- Oracle Binary and Linguistic Sorts in the GDK
- Oracle Language and Character Set Detection in the GDK
- Oracle Translated Locale and Time Zone Names in the GDK
- Using the GDK for E-Mail Programs

**Oracle Locale Information in the GDK**

Oracle locale definitions, which include languages, territories, linguistic sorts, and character sets, are exposed in the GDK Java API. The naming convention that Oracle uses may also be different from other vendors. Although many of these names and definitions follow industry standards, some are Oracle-specific, tailored to meet special customer requirements.

*OraLocaleInfo* is an Oracle locale class that includes language, territory, and collator objects. It provides a method for applications to retrieve a collection of locale-related objects for a given locale, for example, a full list of the Oracle linguistic sorts available in the GDK, the local time zones defined for a given territory, or the common languages used in a particular territory.
The following are examples of using the `OraLocaleInfo` class:

```java
// All Territories supported by GDK
String[] avterr = OraLocaleInfo.getAvailableTerritories();

// Local TimeZones for a given Territory
OraLocaleInfo oloc = OraLocaleInfo.getInstance("English", "Canada");
TimeZone[] loctz = oloc.getLocalTimeZones();
```

**Oracle Locale Mapping in the GDK**

The GDK Java API provides the `LocaleMapper` class. It maps equivalent locales and character sets between Java, IANA, ISO, and Oracle. A Java application may receive locale information from the client that is specified in Oracle’s locale name or an IANA character set name. The Java application must be able to map to an equivalent Java locale or Java encoding before it can process the information correctly.

The following is an example of using the `LocaleMapper` class.

```java
// Mapping from Java locale to Oracle language and Oracle territory
Locale locale = new Locale("it", "IT");
String oraLang = LocaleMapper.getOraLanguage(locale);  
String oraTerr = LocaleMapper.getOraTerritory(locale);

// From Oracle language and Oracle territory to Java Locale
locale = LocaleMapper.getJavaLocale("AMERICAN","AMERICA");
locale = LocaleMapper.getJavaLocale("TRADITIONAL CHINESE", "");

// From IANA & Java to Oracle Character set
String ocs1     = LocaleMapper.getOraCharacterSet(LocaleMapper.IANA, "ISO-8859-1");
String ocs2     = LocaleMapper.getOraCharacterSet(LocaleMapper.JAVA, "ISO8859_1");
```

The `LocaleMapper` class can also return the most commonly used e-mail character set for a specific locale on both Windows and UNIX platforms. This is useful when developing Java applications that need to process e-mail messages.

**See Also:**  "Using the GDK for E-Mail Programs" on page 8-35
Oracle Character Set Conversion (JDK 1.4 and Later) in the GDK

The GDK Java API contains a set of character set conversion classes APIs that enable users to perform Oracle character set conversions. Although Java JDK is already equipped with classes that can perform conversions for many of the standard character sets, they do not support Oracle-specific character sets and Oracle’s user-defined character sets.

In JDK 1.4, J2SE introduced an interface for developers to extend Java's character sets. The GDK Java API provides implicit support for Oracle's character sets by using this plug-in feature. You can access the J2SE API to obtain Oracle-specific behaviors.

Figure 8–5 shows that the GDK character set conversion tables are plugged into J2SE in the same way as the Java character set tables. With this pluggable framework of J2SE, the Oracle character set conversions can be used in the same way as other Java character set conversions.

Figure 8–5 Oracle Character Set Plug-In

Because the java.nio.charset Java package is not available in JDK versions before 1.4, you must install JDK 1.4 or later to use Oracle’s character set plug-in feature.

The GDK character conversion classes support all Oracle character sets including user-defined characters sets. It can be used by Java applications to properly convert to and from Java’s internal character set, UTF-16.

Oracle’s character set names are proprietary. To avoid potential conflicts with Java's own character sets, all Oracle character set names have an X-ORACLE- prefix for all implicit usage through Java’s API.

The following is an example of Oracle character set conversion:
// Converts the Chinese character "three" from UCS2 to JA16SJIS

String str = "\u4e09";
byte[] barr = str.getBytes("x-oracle-JA16SJIS");

Like other Java character sets, the character set facility in
java.nio.charset.Charset is also application to all of the Oracle character
sets. For example, if you wish to check whether the specified character set is a
superset of another character set, then you can use the Charset.contains method as follows:

Charset cs1 = Charset.forName("x-oracle-US7ASCII");
Charset cs2 = Charset.forName("x-oracle-WE8WINDOWS1252");
// true if WE8WINDOWS1252 is the superset of US7ASCII, otherwise false.
boolean osc = cs2.contains(cs1);

For a Java application that is using the JDBC driver to communicate with the
database, the JDBC driver provides the necessary character set conversion between
the application and the database. Calling the GDK character set conversion
methods explicitly within the application is not required. A Java application that
interprets and generates text files based on Oracle’s character set encoding format is
an example of using Oracle character set conversion classes.

Oracle Date, Number, and Monetary Formats in the GDK

The GDK Java API provides formatting classes that support date, number, and
monetary formats using Oracle conventions for Java applications in the
oracle.i18n.text package.

New locale formats introduced in Oracle Database 10g, such as the short and long
date, number, and monetary formats, are also exposed in these format classes.

The following are examples of Oracle date, Oracle number, and Oracle monetary
formatting:

// Obtain the current date and time in the default Oracle LONG format for
// the locale de_DE (German_Germany)

Locale locale = new Locale("de", "DE");
OraDateFormat odf =
    OraDateFormat.getDateTimeInstance(OraDateFormat.LONG, locale);

// Obtain the numeric value 1234567.89 using the default number format
// for the Locale en_IN (English_India)
locale = new Locale("en", "IN");
OraNumberFormat onf = OraNumberFormat.getNumberInstance(locale);
String nm = onf.format(new Double(1234567.89));

// Obtain the monetary value 1234567.89 using the default currency
// format for the Locale en_US (American_America)
locale = new Locale("en", "US");

onf = OraNumberFormat.getCurrencyInstance(locale);
nm = onf.format(new Double(1234567.89));

Oracle Binary and Linguistic Sorts in the GDK

Oracle provides support for binary, monolingual, and multilingual linguistic sorts in the database. In Oracle Database 10g, these sorts have been expanded to provide case-insensitive and accent-insensitive sorting and searching capabilities inside the database. By using the OraCollator class, the GDK Java API enables Java applications to sort and search for information based on the latest Oracle binary and linguistic sorting features, including case-insensitive and accent-insensitive options.

Normalization can be an important part of sorting. The composition and decomposition of characters are based on the Unicode Standard, so sorting also depends on the Unicode standard. Because each version of the JDK may support a different version of the Unicode Standard, the GDK provides an OraNormalizer class based on the Unicode 3.2 standard. It contains methods to perform composition.

The sorting order of a binary sort is based on the Oracle character set that is being used. Except for the UTFE character set, the binary sorts of all Oracle character sets are supported in the GDK Java API. The only linguistic sort that is not supported in the GDK Java API is JAPANESE, but a similar and more accurate sorting result can be achieved by using JAPANESE_M.

The following are examples of string comparisons and string sorting:

// compares strings using XGERMAN

private static String s1 = "abcSS";
private static String s2 = "abc\u00DF";

String cname = "XGERMAN";
OraCollator ocol = OraCollator.getInstance(cname);
int c = ocol.compare(s1, s2);
Oracle Language and Character Set Detection in the GDK

The Oracle Language and Character Set Detection Java classes in the GDK Java API provide a high performance, statistically based engine for determining the character set and language for unspecified text. It can automatically identify language and character set pairs, from throughout the world. With each text, the language and character set detection engine sets up a series of probabilities, each probability corresponding to a language and character set pair. The most probable pair statistically identifies the dominant language and character set.

The purity of the text submitted affects the accuracy of the language and character set detection. Only plain text strings are accepted, so any tagging needs to be stripped before hand. The ideal case is literary text with almost no foreign words or grammatical errors. Text strings that contain a mix of languages or character sets, or
nonnatural language text like addresses, phone numbers, and programming language code may yield poor results.

The LCSDetector class can detect the language and character set of a byte array, a character array, a string, and an InputStream class. It can take the entire input for sampling or only portions of the input for sampling, when the length or both the offset and the length are supplied. For each input, up to three potential language and character set pairs can be returned by the LCSDetector class. They are always ranked in sequence, with the pair with the highest probability returned first.

See Also: "Language and Character Set Detection Support" on page A-23 for a list of supported language and character set pairs

The following are examples of using the LCSDetector class to enable language and character set detection:

```java
// This example detects the character set of a plain text file "foo.txt" and
// then appends the detected ISO character set name to the name of the text file
LCSDetector lc = new LCSDetector();
File oldfile = new File("foo.txt");
FileInputStream in = new FileInputStream(oldfile);
lc.detect(in);
String charset = lc.getResult().getIANACharacterSet();
File newfile = new File("foo."+charset+".txt");
oldfile.renameTo(newfile);
```

```java
// This example shows how to use the LCSDetector class to detect the language and
// character set of a byte array
int offset = 0;
LCSDetector led = new LCSDetector();
/* loop through the entire byte array */
while (true) {
    bytes_read = led.detect(byte_input, offset, 1024);
    if (bytes_read == -1)
        break;
    offset += bytes_read;
}
LCSResultSet res = led.getResult();
/* print the detection results with close ratios */
System.out.println("the best guess ");
```

See Also: "Language and Character Set Detection Support" on page A-23 for a list of supported language and character set pairs
System.out.println("Langauge " + res.getOraLanguage() );
System.out.println("CharacterSet " + res.getOraCharacterSet() );
int high_hit = res.getHiHitPairs();
if ( high_hit >= 2 )
{
    System.out.println("the second best guess ");
    System.out.println("Langauge " + res.getOraLanguage(2) );
    System.out.println("CharacterSet " +res.getOraCharacterSet(2) );
}
if ( high_hit >= 3 )
{
    System.out.println("the third best guess ");
    System.out.println("Langauge " + res.getOraLanguage(3) );
    System.out.println("CharacterSet " +res.getOraCharacterSet(3) );
}

Oracle Translated Locale and Time Zone Names in the GDK

All of the Oracle language names, territory names, character set names, linguistic sort names, and time zone names have been translated into 27 languages including English. They are readily available for inclusion into the user applications, and they provide consistency for the display names across user applications in different languages. OraDisplayLocaleInfo is a utility class that provides the translations of locale and attributes. The translated names are useful for presentation in user interface text and for drop-down selection boxes. For example, a native French speaker prefers to select from a list of time zones displayed in French than in English.

The following is an example of using OraDisplayLocaleInfo to return a list of time zones supported in Canada, using the French translation names:

OraLocaleInfo oloc = OraLocaleInfo.getInstance("CANADIAN FRENCH", "CANADA");
OraDisplayLocaleInfo odloc = OraDisplayLocaleInfo.getInstance(oloc);
TimeZone[] loctzs = oloc.getLocaleTimeZones();
String [] disptz = new string [loctzs.length];
for (int i=0; i<loctzs.length; ++i)
{
    disptz [i]= odloc.getTimeZone(loctzs[i]);
 ...
}
Using the GDK for E-Mail Programs

You can use the GDK LocaleMapper class to retrieve the most commonly used e-mail character set. Call LocaleMapper.getIANACharSetFromLocale, passing in the locale object. The return value is an array of character set names. The first character set returned is the most commonly used e-mail character set.

The following is an example of sending an e-mail message containing Simplified Chinese data in GBK:

```java
import oracle.i18n.util.LocaleMapper;
import java.util.Date;
import java.util.Locale;
import java.util.Properties;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeUtility;
/**
 * Email send operation sample
 *
 * javac -classpath orai18n.jar:j2ee.jar EmailSampleText.java
 * java  -classpath .:orai18n.jar:j2ee.jar EmailSampleText
 */
public class EmailSampleText
{
    public static void main(String[] args)
    {
        send("localhost", // smtp host name
             "your.address@your-company.com", // from email address
             "You", // from display email
             "somebody@some-company.com", // to email address
             "Subject test zh CN", // subject
             "Content "4E02 from Text email", // body
             new Locale("zh", "CN") // user locale
           );
    }
    public static void send(String smtp, String fromEmail, String fromDispName,
                            String toEmail, String subject, String content, Locale locale
                            )
    {
        // get the list of common email character sets
        final String[] charset = LocaleMapper.getIANACharSetFromLocale(LocaleMapper.
```
EMAIL_WINDOWS,
locale
);
// pick the first one for the email encoding
final String contentType = "text/plain; charset=" + charset[0];
try
{
    Properties props = System.getProperties();
    props.put("mail.smtp.host", smtp);
    // here, set username / password if necessary
    Session session = Session.getDefaultInstance(props, null);
    MimeMessage mimeMessage = new MimeMessage(session);
    mimeMessage.setFrom(new InternetAddress(fromEmail, fromDispName, charset[0])
    )
    );
    mimeMessage.setRecipients(Message.RecipientType.TO, toEmail);
    mimeMessage.setSubject(MimeUtility.encodeText(subject, charset[0], "Q"));
    // body
    mimeMessage.setContent(content, contentType);
    mimeMessage.setHeader("Content-Type", contentType);
    mimeMessage.setHeader("Content-Transfer-Encoding", "8bit");
    mimeMessage.setSentDate(new Date());
    Transport.send(mimeMessage);
}
catch (Exception e)
{
    e.printStackTrace();
}

The GDK Application Configuration File

The GDK application configuration file dictates the behavior and the properties of the GDK application framework and the application that is using it. It contains locale mapping tables and parameters for the configuration of the application. One configuration file is required for each application.

The gdkapp.xml application configuration file is a XML document. This file resides in the .WEB-INF directory of the J2EE environment of the application.

The following sections describe the contents and the properties of the application configuration file in detail:
locale-charset-map

This section enables applications to override the mapping from default language to character set provided by the GDK. This mapping is used when the page-charset is set to AUTO-CHARSET.

For example, for the en locale, the default GDK character set is windows-1252. However, if the application requires ISO-8859-1, this can be specified as follows:

```xml
<locale-charset-maps>
  <locale-charset>
    <locale>en</locale>
    <charset>ISO_8859-1</charset>
  </locale-charset>
</locale-charset-maps>
```

The locale name is comprised of the language code and the country code, and they should follow the ISO naming convention as defined in ISO 639 and ISO 3166, respectively. The character set name follows the IANA convention.

Optionally, the user-agent parameter can be specified in the mapping table to distinguish different clients.

```xml
<locale-charset>
  <locale>en,de</locale>
  <user-agent>^Mozilla/4.0$</user-agent>
  <charset>ISO-8859-1</charset>
</locale-charset>
```

The previous example shows that if the user-agent value in the HTTP header starts with Mozilla/4.0 (which indicates older version of web clients) for English (en) and German (de) locales, then the GDK sets the character set to ISO-8859-1.
Multiple locales can be specified in a comma-delimited list.

**See Also:** "page-charset" on page 8-38

### page-charset

This tag section defines the character set of the application pages. If this is explicitly set to a given character set, then all pages use this character set. The character set name must follow the IANA character set convention.

```xml
<page-charset>UTF-8</page-charset>
```

However, if the `page-charset` is set to `AUTO-CHARSET`, then the character set is based on the default character set of the current user locale. The default character set is derived from the locale to character set mapping table specified in the application configuration file.

If the character set mapping table in the application configuration file is not available, then the character set is based on the default locale name to IANA character set mapping table in the GDK. Default mappings are derived from `OraLocaleInfo` class.

**See Also:**
- "locale-charset-map" on page 8-37
- "Handling Non-ASCII Input and Output in the GDK Framework" on page 8-21

### application-locales

This tag section defines a list of the locales supported by the application.

```xml
<application-locales>
  <locale default="yes">en-US</locale>
  <locale>de</locale>
  <locale>zh-CN</locale>
</application-locales>
```

If the language component is specified with the * country code, then all locale names with this language code qualify. For example, if `de-*` (the language code for German) is defined as one of the application locales, then this supports `de-AT` (German-Austria), `de` (German-Germany), `de-LU` (German-Luxembourg), `de-CH` (German-Switzerland), and even irregular locale combination such as `de-CN`
(German-China). However, the application can be restricted to support a predefined set of locales.

It is recommended to set one of the application locales as the default application locale (by specifying `default="yes"`) so that it can be used as a fall back locale for customers who are connecting to the application with an unsupported locale.

**locale-determine-rule**

This section defines the order in which the preferred user locale is determined. The locale sources should be specified based on the scenario in the application. This section includes the following scenarios:

- **Scenario 1:** The GDK framework uses the accept language at all times.
  
  ```xml
  <locale-source>oracle.i18n.servlet.localesource.HTTPAcceptLanguage</locale-source>
  ```

- **Scenario 2:** By default, the GDK framework uses the accept language. After the user specifies the locale, the locale is used for further operations.
  
  ```xml
  <locale-source>oracle.i18n.servlet.localesource.UserInput</locale-source>
  <locale-source>oracle.i18n.servlet.localesource.HTTPAcceptLanguage</locale-source>
  ```

- **Scenario 3:** By default, the GDK framework uses the accept language. After the user is authenticated, the GDK framework uses the database locale source. The database locale source is cached until the user logs out. After the user logs out, the accept language is used again.
  
  ```xml
  <db-locale-source
  data-source-name="jdbc/OracleCoreDS"
  locale-source-table="customer"
  user-column="customer_email"
  user-key="userid"
  language-column="nls_language"
  territory-column="nls_territory"
  timezone-column="timezone"
  >oracle.i18n.servlet.localesource.DBLocaleSource</db-locale-source>
  <locale-source>oracle.i18n.servlet.localesource.HttpAcceptLanguage</locale-source>
  ```

Note that Scenario 3 includes the predefined database locale source, DBLocaleSource. It enables the user profile information to be specified in the configuration file without writing a custom database locale source. In the example, the user profile table is called "customer". The columns are "customer_email", "nls_language", "nls_territory", and "timezone". They store the unique e-mail address, the Oracle name of the preferred language, the Oracle name of the preferred territory, and the time
The GDK Application Configuration File

zone ID of a customer. The user-key is a mandatory attribute that specifies the attribute name used to pass the user ID from the application to the GDK framework.

Scenario 4: The GDK framework uses the accept language in the first page. When the user inputs a locale, it is cached and used until the user logs into the application. After the user is authenticated, the GDK framework uses the database locale source. The database locale source is cached until the user logs out. After the user logs out, the accept language is used again or the user input is used if the user inputs a locale.

<locale-source>demo.DatabaseLocaleSource</locale-source>
<locale-source>oracle.i18n.servlet.localesource.UserInput</locale-source>
<locale-source>oracle.i18n.servlet.localesource.HttpAcceptLanguage</locale-source>

Note that Scenario 4 uses the custom database locale source. If the user profile schema is complex, such as user profile information separated into multiple tables, then the custom locale source should be provided by the application. Examples of custom locale sources can be found in the $ORACLE_HOME/nls/gdk/demo directory.

locale-parameter-name

The tag defines the name of the locale parameters that are used in the user input so that the current user locale can be passed between requests.

Table 8–3 shows the parameters used in the GDK framework.

<table>
<thead>
<tr>
<th>Default Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>locale</td>
<td>ISO locale where ISO 639 language code and ISO 3166 country code are connected with an underscore (_) or a hyphen (-). For example, zh_CN for Simplified Chinese used in China</td>
</tr>
<tr>
<td>language</td>
<td>Oracle language name. For example, AMERICAN for American English</td>
</tr>
<tr>
<td>territory</td>
<td>Oracle territory name. For example, SPAIN</td>
</tr>
<tr>
<td>timezone</td>
<td>Timezone name. For example, America/Los_Angeles</td>
</tr>
<tr>
<td>iso-currency</td>
<td>ISO 4217 currency code. For example, EUR for the euro</td>
</tr>
<tr>
<td>date-format</td>
<td>Date format pattern mask. For example, DD_MON_RRRR</td>
</tr>
<tr>
<td>long-date-format</td>
<td>Long date format pattern mask. For example, DAY-YYY-MM-DD</td>
</tr>
<tr>
<td>date-time-format</td>
<td>Date and time format pattern mask. For example, DD-MON-RRRR HH24:MI:SS</td>
</tr>
</tbody>
</table>
The GDK Application Configuration File

Table 8–3  Locale Parameters Used in the GDK Framework (Cont.)

<table>
<thead>
<tr>
<th>Default Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>long-date-time-format</td>
<td>Long date and time format pattern mask. For example, <code>DAY YYYY-MM-DD HH12:MI:SS AM</code></td>
</tr>
<tr>
<td>time-format</td>
<td>Time format pattern mask. For example, <code>HH:MI:SS</code></td>
</tr>
<tr>
<td>number-format</td>
<td>Number format. For example, <code>9G99G990D00</code></td>
</tr>
<tr>
<td>currency-format</td>
<td>Currency format. For example, <code>L9G99G990D00</code></td>
</tr>
<tr>
<td>linguistic-sorting</td>
<td>Linguistic sort order name. For example, <code>JAPANESE_M</code> for Japanese multilingual sort</td>
</tr>
<tr>
<td>charset</td>
<td>Character set. For example, <code>WE8ISO8859P15</code></td>
</tr>
<tr>
<td>writing-direction</td>
<td>Writing direction string. For example, <code>LTR</code> for left-to-right writing direction or <code>RTL</code> for right-to-left writing direction</td>
</tr>
<tr>
<td>command</td>
<td>GDK command. For example, <code>store</code> for the update operation</td>
</tr>
</tbody>
</table>

The parameter names are used in either the parameter in the HTML form or in the URL.

**message-bundles**

This tag defines the base class names of the resource bundles used in the application. The mapping is used in the `Localizer.getMessage` method for locating translated text in the resource bundles.

```xml
<message-bundles>
  <resource-bundle>Messages</resource-bundle>
  <resource-bundle name="newresource">NewMessages</resource-bundle>
</message-bundles>
```

If the name attribute is not specified or if it is specified as `name="default"` to the `<resource-bundle>` tag, then the corresponding resource bundle is used as the default message bundle. To support more than one resource bundle in an application, resource bundle names must be assigned to the nondefault resource bundles. The nondefault bundle names must be passed as a parameter of the `getMessage` method.

For example:

```java
Localizer loc = ServletHelper.getLocalizerInstance(request);
String translatedMessage = loc.getMessage("Hello");
```
String translatedMessage2 = loc.getMessage("World", "newresource");

**url-rewrite-rule**

This tag is used to control the behavior of the URL rewrite operations. The rewriting rule is a regular expression.

```
<url-rewrite-rule fallback="no">
  <pattern>(.*)/([^/]+)$</pattern>
  <result>$1/$L/$2</result>
</url-rewrite-rule>
```

**See Also:** "Managing Localized Content in the GDK" on page 8-24

If the localized content for the requested locale is not available, then it is possible for the GDK framework to trigger the locale fallback mechanism by mapping it to the closest translation locale. By default the fallback option is turned off. This can be turned on by specifying `fallback="yes"`.

For example, suppose an application supports only the following translations: en, de, and ja, and en is the default locale of the application. If the current application locale is de-US, then it falls back to de. If the user selects zh-TW as its application locale, then it falls back to en.

A fallback mechanism is often necessary if the number of supported application locales is greater than the number of the translation locales. This usually happens if multiple locales share one translation. One example is Spanish. The application may need to support multiple Spanish-speaking countries and not just Spain, with one set of translation files.

Multiple URL rewrite rules can be specified by assigning the name attribute to nondefault URL rewrite rules. To use the nondefault URL rewrite rules, the name must be passed as a parameter of the rewrite URL method. For example:

```
<img src="<%=ServletHelper.rewriteURL("images/welcome.gif", request) %>>
<img src="<%=ServletHelper.rewriteURL("US.gif", "flag", request) %>>
```

The first rule changes the "images/welcome.gif" URL to the localized welcome image file. The second rule named "flag" changes the "US.gif" URL to the user's country flag image file. The rule definition should be as follows:

```
<url-rewrite-rule fallback="yes">
  <pattern>(.*)/([^/]+)$</pattern>
  <result>$1/$L/$2</result>
</url-rewrite-rule>
```
Example: GDK Application Configuration File

This section contains an example of an application configuration file with the following application properties:

- The application supports the following locales: Arabic (ar), Greek (el), English (en), German (de), French (fr), Japanese (ja) and Simplified Chinese for China (zh-CN).
- English is the default application locale.
- The page character set for the ja locale is always UTF-8.
- The page character set for the en and de locales when using an Internet Explorer client is windows-1252.
- The page character set for the en, de, and fr locales on other web browser clients is iso-8859-1.
- The page character set for all other locales are the default character set for the locale.
- The user locale is determined by the following order: user input locale and then Accept-Language.
- The localized contents are stored in their appropriate language subfolders. The folder names are derived from the ISO 639 language code. The folders are located in the root directory of the application. For example, the Japanese file for /shop/welcome.jpg is stored in /ja/shop/welcome.jpg.

```xml
<xml version="1.0" encoding="utf-8"?>
<gdkapp
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="gdkapp.xsd">
 <!-- Language to Character set mapping -->
 <locale-charset-maps>
 <locale-charset>
 <locale>ja</locale>
 <charset>UTF-8</charset>
 </locale-charset>
 <locale-charset>
 <locale>en, de</locale>
 <charset>UTF-8</charset>
 </locale-charset>
 <locale-charset>
 <locale>en, de</locale>
 <user-agent>"Mozilla/\[0-9\].+\{compatible; MSIE \^[;]+; \}</user-agent>
</gdkapp>
```
GDK for Java Supplied Packages and Classes

Oracle Globalization Services for Java contains the following packages:

- `oracle.i18n.lcsd`
- `oracle.i18n.net`
- `oracle.i18n.servlet`
- `oracle.i18n.text`
- `oracle.i18n.util`

See Also: Oracle Globalization Development Kit Java API Reference
oracle.i18n.lcsd

Package oracle.i18n.lcsd provides classes to automatically detect and recognize language and character set based on text input. Language is based on ISO; encoding is based on IANA or Oracle character sets. It includes the following classes:

- **LCSDetector**: Contains methods to automatically detect and recognize language and character set based on text input.
- **LCSDResultSet**: The LCSDResultSet class is for storing the result generated by LCSDetector. Methods in this class can be used to retrieve specific information from the result.

oracle.i18n.net

Package oracle.i18n.net provides Internet-related data conversions for globalization. It includes the following classes:

- **CharEntityReference**: A utility class to escape or unescape a string into character reference or entity reference form
- **CharEntityReference.Form**: A form parameter class that specifies the escaped form

oracle.i18n.servlet

Package oracle.i18n.Servlet enables JSP and JavaServlet to have automatic locale support and also returns the localized contents to the application. It includes the following classes:

- **ApplicationContext**: An application context class that governs application scope operation in the framework
- **Localizer**: An all-in-one object class that enables access to the most commonly used globalization information
- **ServletHelper**: A delegate class that bridges between Java servlets and globalization objects

oracle.i18n.text

Package oracle.i18n.text provides general text data globalization support. It includes the following classes:
GDK for PL/SQL Supplied Packages

- **OraCollationKey**: A class which represents a string under certain rules of a specific *OraCollator* object
- **OraCollator**: A class to perform locale-sensitive string comparison, including linguistic collation and binary sorting
- **OraDateFormat**: An abstract class to do formatting and parsing between datetime and string locale. It supports Oracle datetime formatting behavior.
- **OraDecimalFormat**: A concrete class to do formatting and parsing between number and string locale. It supports Oracle number formatting behavior.
- **OraDecimalFormatSymbol**: A class to maintain Oracle format symbols used by Oracle number and currency formatting
- **OraNumberFormat**: An abstract class to do formatting and parsing between number and string locale. It supports Oracle number formatting behavior.
- **OraSimpleDateFormat**: A concrete class to do formatting and parsing between datetime and string locale. It supports Oracle datetime formatting behavior.

**oracle.i18n.util**

Package *oracle.i18n.util* provides general utilities for globalization support. It includes the following classes:
- **LocaleMapper**: Provides mappings between Oracle locale elements and equivalent locale elements in other vendors and standards
- **OraDisplayLocaleInfo**: A translation utility class that provides the translations of locale and attributes
- **OraLocaleInfo**: An Oracle locale class that includes the language, territory, and collator objects
- **OraSQLUtil**: An Oracle SQL Utility class that includes some useful methods of dealing with SQL

**GDK for PL/SQL Supplied Packages**

The GDK for PL/SQL includes the following PL/SQL packages:
- **UTL_I18N**
- **UTL_LMS**
**GDK Error Messages**

**UTL_I18N** is a set of PL/SQL services that help developers to build globalized applications. The **UTL_I18N** PL/SQL package provides the following functions:

- String conversion functions for various datatypes
- Escape and unescape sequences for predefined characters and multibyte characters used by HTML and XML documents
- Functions that map between Oracle, Internet Assigned Numbers Authority (IANA), ISO, and e-mail application character sets, languages, and territories
- A function that returns the Oracle character set name from an Oracle language name

**UTL_LMS** retrieves and formats error messages in different languages.

**See Also:**  *PL/SQL Packages and Types Reference*

---

**GDK Error Messages**

**GDK-03001 Invalid or unsupported sorting rule**

*Cause:*  An invalid or unsupported sorting rule name was specified.

*Action:*  Choose a valid sorting rule name and check the Globalization Support Guide for the list of sorting rule names.

**GDK-03002 The functional-driven sort is not supported.**

*Cause:*  A functional-driven sorting rule name was specified.

*Action:*  Choose a valid sorting rule name and check the Globalization Support Guide for the list of sorting rule names.

**GDK-03003 The linguistic data file is missing.**

*Cause:*  A valid sorting rule was specified, but the associated data file was not found.

*Action:*  Make sure the GDK jar files are correctly installed in the Java application.

**GDK-03005 Binary sort is not available for the specified character set.**

*Cause:*  Binary sorting for the specified character set is not supported.

*Action:*  Check the Globalization Support Guide for a character set that supports binary sort.

**GDK-03006 The comparison strength level setting is invalid.**
GDK Error Messages

**Cause:** An invalid comparison strength level was specified.

**Action:** Choose a valid comparison strength level from the list -- PRIMARY, SECONDARY or TERTIARY.

**GDK-03007 The composition level setting is invalid.**

**Cause:** An invalid composition level setting was specified.

**Action:** Choose a valid composition level from the list -- NO_COMPOSITION or CANONICAL_COMPOSITION.

**GDK-04001 Cannot map Oracle character to Unicode**

**Cause:** The program attempted to use a character in the Oracle character set that cannot be mapped to Unicode.

**Action:** Write a separate exception handler for the invalid character, or call the withReplacement method so that the invalid character can be replaced with a valid replacement character.

**GDK-04002 Cannot map Unicode to Oracle character**

**Cause:** The program attempted to use a Unicode character that cannot be mapped to a character in the Oracle character set.

**Action:** Write a separate exception handler for the invalid character, or call the withReplacement method so that the invalid character can be replaced with a valid replacement character.

**GDK-05000 A literal in the date format is too large.**

**Cause:** The specified string literal in the date format was too long.

**Action:** Use a shorter string literal in the date format.

**GDK-05001 The date format is too long for internal buffer.**

**Cause:** The date format pattern was too long.

**Action:** Use a shorter date format pattern.

**GDK-05002 The Julian date is out of range.**

**Cause:** An illegal date range was specified.

**Action:** Make sure that date is in the specified range 0 - 3439760.

**GDK-05003 Failure in retrieving date/time**

**Cause:** This is an internal error.

**Action:** Contact Oracle Support Services.
GDK-05010 Duplicate format code found
  Cause: The same format code was used more than once in the format pattern.
  Action: Remove the redundant format code.

GDK-05011 The Julian date precludes the use of the day of the year.
  Cause: Both the Julian date and the day of the year were specified.
  Action: Remove either the Julian date or the day of the year.

GDK-05012 The year may only be specified once.
  Cause: The year format code appeared more than once.
  Action: Remove the redundant year format code.

GDK-05013 The hour may only be specified once.
  Cause: The hour format code appeared more than once.
  Action: Remove the redundant hour format code.

GDK-05014 The AM/PM conflicts with the use of A.M./P.M.
  Cause: AM/PM was specified along with A.M./P.M.
  Action: Use either AM/PM or A.M./P.M; do not use both.

GDK-05015 The BC/AD conflicts with the use of B.C./A.D.
  Cause: BC/AD was specified along with B.C./A.D.
  Action: Use either BC/AD or B.C./A.D.; do not use both.

GDK-05016 Duplicate month found
  Cause: The month format code appeared more than once.
  Action: Remove the redundant month format code.

GDK-05017 The day of the week may only be specified once.
  Cause: The day of the week format code appeared more than once.
  Action: Remove the redundant day of the week format code.

GDK-05018 The HH24 precludes the use of meridian indicator.
  Cause: HH24 was specified along with the meridian indicator.
  Action: Use either the HH24 or the HH12 with the meridian indicator.

GDK-05019 The signed year precludes the use of BC/AD.
  Cause: The signed year was specified along with BC/AD.
Action: Use either the signed year or the unsigned year with BC/AD.

GDK-05020 A format code cannot appear in a date input format.
Cause: A format code appeared in a date input format.
Action: Remove the format code.

GDK-05021 Date format not recognized
Cause: An unsupported format code was specified.
Action: Correct the format code.

GDK-05022 The era format code is not valid with this calendar.
Cause: An invalid era format code was specified for the calendar.
Action: Remove the era format code or use another calendar that supports the era.

GDK-05030 The date format pattern ends before converting entire input string.
Cause: An incomplete date format pattern was specified.
Action: Rewrite the format pattern to cover the entire input string.

GDK-05031 The year conflicts with the Julian date.
Cause: An incompatible year was specified for the Julian date.
Action: Make sure that the Julian date and the year are not in conflict.

GDK-05032 The day of the year conflicts with the Julian date.
Cause: An incompatible day of year was specified for the Julian date.
Action: Make sure that the Julian date and the day of the year are not in conflict.

GDK-05033 The month conflicts with the Julian date.
Cause: An incompatible month was specified for the Julian date.
Action: Make sure that the Julian date and the month are not in conflict.

GDK-05034 The day of the month conflicts with the Julian date.
Cause: An incompatible day of the month was specified for the Julian date.
Action: Make sure that the Julian date and the day of the month are not in conflict.

GDK-05035 The day of the week conflicts with the Julian date.
Cause: An incompatible day of the week was specified for the Julian date.
Action: Make sure that the Julian date and the day of week are not in conflict.

GDK-05036 The hour conflicts with the seconds in the day.
Cause: The specified hour and the seconds in the day were not compatible.
Action: Make sure the hour and the seconds in the day are not in conflict.

GDK-05037 The minutes of the hour conflicts with the seconds in the day.
Cause: The specified minutes of the hour and the seconds in the day were not compatible.
Action: Make sure the minutes of the hour and the seconds in the day are not in conflict.

GDK-05038 The seconds of the minute conflicts with the seconds in the day.
Cause: The specified seconds of the minute and the seconds in the day were not compatible.
Action: Make sure the seconds of the minute and the seconds in the day are not in conflict.

GDK-05039 Date not valid for the month specified
Cause: An illegal date for the month was specified.
Action: Check the date range for the month.

GDK-05040 Input value not long enough for the date format
Cause: Too many format codes were specified.
Action: Remove unused format codes or specify a longer value.

GDK-05041 A full year must be between -4713 and +9999, and not be 0.
Cause: An illegal year was specified.
Action: Specify the year in the specified range.

GDK-05042 A quarter must be between 1 and 4.
Cause: An illegal quarter was specified.
Action: Make sure that the quarter is in the specified range.

GDK-05043 Not a valid month
Cause: An illegal month was specified.
Action: Make sure that the month is between 1 and 12 or has a valid month name.
GDK-05044 The week of the year must be between 1 and 52.
   Cause: An illegal week of the year was specified.
   Action: Make sure that the week of the year is in the specified range.

GDK-05045 The week of the month must be between 1 and 5.
   Cause: An illegal week of the month was specified.
   Action: Make sure that the week of the month is in the specified range.

GDK-05046 Not a valid day of the week
   Cause: An illegal day of the week was specified.
   Action: Make sure that the day of the week is between 1 and 7 or has a valid
day name.

GDK-05047 A day of the month must be between 1 and the last day of the month.
   Cause: An illegal day of the month was specified.
   Action: Make sure that the day of the month is in the specified range.

GDK-05048 A day of year must be between 1 and 365 (366 for leap year).
   Cause: An illegal day of the year was specified.
   Action: Make sure that the day of the year is in the specified range.

GDK-05049 An hour must be between 1 and 12.
   Cause: An illegal hour was specified.
   Action: Make sure that the hour is in the specified range.

GDK-05050 An hour must be between 0 and 23.
   Cause: An illegal hour was specified.
   Action: Make sure that the hour is in the specified range.

GDK-05051 A minute must be between 0 and 59.
   Cause: An illegal minute was specified.
   Action: Make sure the minute is in the specified range.

GDK-05052 A second must be between 0 and 59.
   Cause: An illegal second was specified.
   Action: Make sure the second is in the specified range.

GDK-05053 A second in the day must be between 0 and 86399.
Cause: An illegal second in the day was specified.
Action: Make sure second in the day is in the specified range.

GDK-05054 The Julian date must be between 1 and 5373484.
Cause: An illegal Julian date was specified.
Action: Make sure that the Julian date is in the specified range.

GDK-05055 Missing AM/A.M. or PM/P.M.
Cause: Neither AM/A.M. nor PM/P.M. was specified in the format pattern.
Action: Specify either AM/A.M. or PM/P.M.

GDK-05056 Missing BC/B.C. or AD/A.D.
Cause: Neither BC/B.C. nor AD/A.D. was specified in the format pattern.
Action: Specify either BC/B.C. or AD/A.D.

GDK-05057 Not a valid time zone
Cause: An illegal time zone was specified.
Action: Specify a valid time zone.

GDK-05058 Non-numeric character found
Cause: A non-numeric character was found where a numeric character was expected.
Action: Make sure that the character is a numeric character.

GDK-05059 Non-alphabetic character found
Cause: A non-alphabetic character was found where an alphabetic was expected.
Action: Make sure that the character is an alphabetic character.

GDK-05060 The week of the year must be between 1 and 53.
Cause: An illegal week of the year was specified.
Action: Make sure that the week of the year is in the specified range.

GDK-05061 The literal does not match the format string.
Cause: The string literals in the input were not the same length as the literals in the format pattern (with the exception of the leading whitespace).
Action: Correct the format pattern to match the literal. If the "FX" modifier has been toggled on, the literal must match exactly, with no extra whitespace.
GDK-05062 The numeric value does not match the length of the format item.
   Cause: The numeric value did not match the length of the format item.
   Action: Correct the input date or turn off the FX or FM format modifier. When
   the FX and FM format codes are specified for an input date, then the number of
digits must be exactly the number specified by the format code. For example, 9
will not match the format code DD but 09 will.

GDK-05063 The year is not supported for the current calendar.
   Cause: An unsupported year for the current calendar was specified.
   Action: Check the Globalization Support Guide to find out what years are
supported for the current calendar.

GDK-05064 The date is out of range for the calendar.
   Cause: The specified date was out of range for the calendar.
   Action: Specify a date that is legal for the calendar.

GDK-05065 Invalid era
   Cause: An illegal era was specified.
   Action: Make sure that the era is valid.

GDK-05066 The datetime class is invalid.
   Cause: This is an internal error.
   Action: Contact Oracle Support Services.

GDK-05067 The interval is invalid.
   Cause: An invalid interval was specified.
   Action: Specify a valid interval.

GDK-05068 The leading precision of the interval is too small.
   Cause: The specified leading precision of the interval was too small to store the
interval.
   Action: Increase the leading precision of the interval or specify an interval with
a smaller leading precision.

GDK-05069 Reserved for future use
   Cause: Reserved.
   Action: Reserved.

GDK-05070 The specified intervals and datetimes were not mutually comparable.
GDK Error Messages

Cause: The specified intervals and datetimes were not mutually comparable.
Action: Specify a pair of intervals or datetimes that are mutually comparable.

GDK-05071 The number of seconds must be less than 60.
Cause: The specified number of seconds was greater than 59.
Action: Specify a value for the seconds to 59 or smaller.

GDK-05072 Reserved for future use
Cause: Reserved.
Action: Reserved.

GDK-05073 The leading precision of the interval was too small.
Cause: The specified leading precision of the interval was too small to store the interval.
Action: Increase the leading precision of the interval or specify an interval with a smaller leading precision.

GDK-05074 An invalid time zone hour was specified.
Cause: The hour in the time zone must be between -12 and 13.
Action: Specify a time zone hour between -12 and 13.

GDK-05075 An invalid time zone minute was specified.
Cause: The minute in the time zone must be between 0 and 59.
Action: Specify a time zone minute between 0 and 59.

GDK-05076 An invalid year was specified.
Cause: A year must be at least -4713.
Action: Specify a year that is greater than or equal to -4713.

GDK-05077 The string is too long for the internal buffer.
Cause: This is an internal error.
Action: Contact Oracle Support Services.

GDK-05078 The specified field was not found in the datetime or interval.
Cause: The specified field was not found in the datetime or interval.
Action: Make sure that the specified field is in the datetime or interval.

GDK-05079 An invalid hh25 field was specified.
GDK Error Messages

Cause: The hh25 field must be between 0 and 24.
Action: Specify an hh25 field between 0 and 24.

GDK-05080 An invalid fractional second was specified.
   Cause: The fractional second must be between 0 and 999999999.
   Action: Specify a value for fractional second between 0 and 999999999.

GDK-05081 An invalid time zone region ID was specified.
   Cause: The time zone region ID specified was invalid.
   Action: Contact Oracle Support Services.

GDK-05082 Time zone region name not found
   Cause: The specified region name cannot be found.
   Action: Contact Oracle Support Services.

GDK-05083 Reserved for future use
   Cause: Reserved.
   Action: Reserved.

GDK-05084 Internal formatting error
   Cause: This is an internal error.
   Action: Contact Oracle Support Services.

GDK-05085 Invalid object type
   Cause: An illegal object type was specified.
   Action: Use a supported object type.

GDK-05086 Invalid date format style
   Cause: An illegal format style was specified.
   Action: Choose a valid format style.

GDK-05087 A null format pattern was specified.
   Cause: The format pattern cannot be null.
   Action: Provide a valid format pattern.

GDK-05088 Invalid number format model
   Cause: An illegal number format code was specified.
   Action: Correct the number format code.
GDK-05089 Invalid number
   Cause: An invalid number was specified.
   Action: Correct the input.

GDK-05090 Reserved for future use
   Cause: Reserved.
   Action: Reserved.

GDK-05091 Datetime/interval internal error
   Cause: This is an internal error.
   Action: Contact Oracle Support Services.

GDK-05098 Too many precision specifiers
   Cause: Extra data was found in the date format pattern while the program
       attempted to truncate or round dates.
   Action: Check the syntax of the date format pattern.

GDK-05099 Bad precision specifier
   Cause: An illegal precision specifier was specified.
   Action: Use a valid precision specifier.

GDK-05200 Missing WE8ISO8859P1 data file
   Cause: The character set data file for WE8ISO8859P1 was not installed.
   Action: Make sure the GDK jar files are installed properly in the Java
       application.

GDK-05201 Failed to convert to a hexadecimal value
   Cause: An invalid hexadecimal string was included in the HTML/XML data.
   Action: Make sure the string includes the hexadecimal character in the form of
       &x[0-9A-Fa-f]+;.

GDK-05202 Failed to convert to a decimal value
   Cause: An invalid decimal string was found in the HTML/XML data.
   Action: Make sure the string includes the decimal character in the form of
       &[0-9]+;.

GDK-05203 Unregistered character entity
   Cause: An invalid character entity was found in the HTML/XML data.
**Action:** Use a valid character entity value in HTML/XML data. See HTML/XML standards for the registered character entities.

**GDK-05204 Invalid Quoted-Printable value**

**Cause:** An invalid Quoted-Printable data was found in the data.

**Action:** Make sure the input data has been encoded in the proper Quoted-Printable form.

**GDK-05205 Invalid MIME header format**

**Cause:** An invalid MIME header format was specified.

**Action:** Check RFC 2047 for the MIME header format. Make sure the input data conforms to the format.

**GDK-05206 Invalid numeric string**

**Cause:** An invalid character in the form of %FF was found when a URL was being decoded.

**Action:** Make sure the input URL string is valid and has been encoded correctly; %FF needs to be a valid hex number.

**GDK-05207 Invalid class of the object, key, in the user-defined locale to charset mapping**

**Cause:** The class of key object in the user-defined locale to character set mapping table was not java.util.Locale.

**Action:** When you construct the Map object for the user-defined locale to character set mapping table, specify java.util.Locale for the key object.

**GDK-05208 Invalid class of the object, value, in the user-defined locale to charset mapping**

**Cause:** The class of value object in the user-defined locale to character set mapping table was not java.lang.String.

**Action:** When you construct the Map object for the user-defined locale to character set mapping table, specify java.lang.String for the value object.

**GDK-05209 Invalid rewrite rule**

**Cause:** An invalid regular expression was specified for the match pattern in the rewrite rule.

**Action:** Make sure the match pattern for the rewriting rule uses a valid regular expression.
GDK-05210 Invalid character set
  **Cause:** An invalid character set name was specified.
  **Action:** Specify a valid character set name.

GDK-0521 Default locale not defined as a supported locale
  **Cause:** The default application locale was not included in the supported locale list.
  **Action:** Include the default application locale in the supported locale list or change the default locale to the one that is in the list of the supported locales.

GDK-05212 The rewriting rule must be a String array with three elements.
  **Cause:** The rewriting rule parameter was not a String array with three elements.
  **Action:** Make sure the rewriting rule parameter is a String array with three elements. The first element represents the match pattern in the regular expression, the second element represents the result pattern in the form specified in the JavaDoc of ServletHelper.rewriteURL, and the third element represents the Boolean value "True" or "False" that specifies whether the locale fallback operation is performed or not.

GDK-05213 Invalid type for the class of the object, key, in the user-defined parameter name mapping
  **Cause:** The class of key object in the user-defined parameter name mapping table was not java.lang.String.
  **Action:** When you construct the Map object for the user-defined parameter name mapping table, specify java.lang.String for the key object.

GDK-05214 The class of the object, value, in the user-defined parameter name mapping, must be of type "java.lang.String".
  **Cause:** The class of value object in the user-defined parameter name mapping table was not java.lang.String.
  **Action:** When you construct the Map object for the user-defined parameter name mapping table, specify java.lang.String for the value object.

GDK-05215 Parameter name must be in the form [a-z][a-z0-9]*.
  **Cause:** An invalid character was included in the parameter name.
  **Action:** Make sure the parameter name is in the form of [a-z][a-z0-9]*.
GDK-05216 The attribute "var" must be specified if the attribute "scope" is set.
  Cause: Despite the attribute "scope" being set in the tag, the attribute "var" was not specified.
  Action: Specify the attribute "var" for the name of variable.

GDK-05217 The "param" tag must be nested inside a "message" tag.
  Cause: The "param" tag was not nested inside a "message" tag.
  Action: Make sure the tag "param" is inside the tag "message".

GDK-05218 Invalid "scope" attribute is specified.
  Cause: An invalid "scope" value was specified.
  Action: Specify a valid scope as either "application," "session," "request," or "page".

GDK-05219 Invalid date format style
  Cause: The specified date format style was invalid.
  Action: Specify a valid date format style as either "default," "short," or "long".

GDK-05220 No corresponding Oracle character set exists for the IANA character set.
  Cause: An unsupported IANA character set name was specified.
  Action: Specify the IANA character set that has a corresponding Oracle character set.

GDK-05221 Invalid parameter name
  Cause: An invalid parameter name was specified in the user-defined parameter mapping table.
  Action: Make sure the specified parameter name is supported. To get the list of supported parameter names, call LocaleSource.Parameter.toArray.

GDK-05222 Invalid type for the class of the object, key, in the user-defined message bundle mapping.
  Cause: The class of key object in the user-defined message bundle mapping table was not "java.lang.String."
  Action: When you construct the Map object for the user-defined message bundle mapping table, specify java.lang.String for the key object.
GDK-05223 Invalid type for the class of the object, value, in the user-defined message bundle mapping
   Cause: The class of value object in the user-defined message bundle mapping table was not "java.lang.String."
   Action: When you construct the Map object for the user-defined message bundle mapping table, specify java.lang.String for the value object.

GDK-05224 Invalid locale string
   Cause: An invalid character was included in the specified ISO locale names in the GDK application configuration file.
   Action: Make sure the ISO locale names include only valid characters. A typical name format is an ISO 639 language followed by an ISO 3166 country connected by a dash character; for example, "en-US" is used to specify the locale for American English in the United States.

GDK-06001 LCSDetector profile not available
   Cause: The specified profile was not found.
   Action: Make sure the GDK jar files are installed properly in the Java application.

GDK-06002 Invalid IANA character set name or no corresponding Oracle name found
   Cause: The IANA character set specified was either invalid or did not have a corresponding Oracle character set.
   Action: Check that the IANA character is valid and make sure that it has a corresponding Oracle character set.

GDK-06003 Invalid ISO language name or no corresponding Oracle name found
   Cause: The ISO language specified was either invalid or did not have a corresponding Oracle language.
   Action: Check to see that the ISO language specified is either invalid or does not have a corresponding Oracle language.

GDK-06004 A character set filter and a language filter cannot be set at the same time.
   Cause: A character set filter and a language filter were set at the same time in a LCSDetector object.
   Action: Set only one of the two -- character set or language.
GDK-06005 Reset is necessary before LCSDetector can work with a different data source.

Cause: The reset method was not invoked before a different type of data source was used for a LCSDetector object.

Action: Call LCSDetector.reset to reset the detector before switching to detect other types of data source.

ORA-17154 Cannot map Oracle character to Unicode

Cause: The Oracle character was either invalid or incomplete and could not be mapped to an Unicode value.

Action: Write a separate exception handler for the invalid character, or call the withReplacement method so that the invalid character can be replaced with a valid replacement character.

ORA-17155 Cannot map Unicode to Oracle character

Cause: The Unicode character did not have a counterpart in the Oracle character set.

Action: Write a separate exception handler for the invalid character, or call the withReplacement method so that the invalid character can be replaced with a valid replacement character.
This chapter contains information useful for SQL programming in a globalization support environment. It includes the following topics:

- Locale-Dependent SQL Functions with Optional NLS Parameters
- Other Locale-Dependent SQL Functions
- Miscellaneous Topics for SQL and PL/SQL Programming in a Global Environment
Locale-Dependent SQL Functions with Optional NLS Parameters

All SQL functions whose behavior depends on globalization support conventions allow NLS parameters to be specified. These functions are:

- `TO_CHAR`
- `TO_DATE`
- `TO_NUMBER`
- `NLS_UPPER`
- `NLS_LOWER`
- `NLS_INITCAP`
- `NLSSORT`

Explicitly specifying the optional NLS parameters for these functions enables the functions to be evaluated independently of the session’s NLS parameters. This feature can be important for SQL statements that contain numbers and dates as string literals.

For example, the following query is evaluated correctly if the language specified for dates is AMERICAN:

```
SELECT last_name FROM employees WHERE hire_date > '01-JAN-1999';
```

Such a query can be made independent of the current date language by using a statement similar to the following:

```
SELECT last_name FROM employees
WHERE hire_date > TO_DATE('01-JAN-1999','DD-MON-YYYY',
  'NLS_DATE_LANGUAGE = AMERICAN');
```

In this way, SQL statements that are independent of the session language can be defined where necessary. Such statements are necessary when string literals appear in SQL statements in views, CHECK constraints, or triggers.

**Note:** Only SQL statements that must be independent of the session NLS parameter values should explicitly specify optional NLS parameters in locale-dependent SQL functions. Using session default values for NLS parameters in SQL functions usually results in better performance.

All character functions support both single-byte and multibyte characters. Except where explicitly stated, character functions operate character by character, rather than byte by byte.
The rest of this section includes the following topics:

- Default Values for NLS Parameters in SQL Functions
- Specifying NLS Parameters in SQL Functions
- Unacceptable NLS Parameters in SQL Functions

**Default Values for NLS Parameters in SQL Functions**

When SQL functions evaluate views and triggers, default values from the current session are used for the NLS function parameters. When SQL functions evaluate CHECK constraints, they use the default values that were specified for the NLS parameters when the database was created.

**Specifying NLS Parameters in SQL Functions**

NLS parameters are specified in SQL functions as follows:

'parameter = value'

For example:

'NLS_DATE_LANGUAGE = AMERICAN'

The following NLS parameters can be specified in SQL functions:

- NLS_DATE_LANGUAGE
- NLS_NUMERIC_CHARACTERS
- NLS_CURRENCY
- NLS_ISO_CURRENCY
- NLS_DUAL_CURRENCY
- NLSCALENDAR
- NLS_SORT

Table 9–1 shows which NLS parameters are valid for specific SQL functions.

<table>
<thead>
<tr>
<th>SQL Function</th>
<th>Valid NLS Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>TO_DATE</td>
<td>NLS_DATE_LANGUAGE</td>
</tr>
<tr>
<td></td>
<td>NLSCALENDAR</td>
</tr>
</tbody>
</table>

Table 9–1  SQL Functions and Their Valid NLS Parameters
Locale-Dependent SQL Functions with Optional NLS Parameters

Table 9–1  SQL Functions and Their Valid NLS Parameters (Cont.)

<table>
<thead>
<tr>
<th>SQL Function</th>
<th>Valid NLS Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>TO_NUMBER</td>
<td>NLS_NUMERIC_CHARACTERS</td>
</tr>
<tr>
<td></td>
<td>NLS.Currency</td>
</tr>
<tr>
<td></td>
<td>NLS_DUAL_CURRENCY</td>
</tr>
<tr>
<td></td>
<td>NLS_ISO_CURRENCY</td>
</tr>
<tr>
<td>TO_CHAR</td>
<td>NLS_DATE_LANGUAGE</td>
</tr>
<tr>
<td></td>
<td>NLS_NUMERIC_CHARACTERS</td>
</tr>
<tr>
<td></td>
<td>NLS.Currency</td>
</tr>
<tr>
<td></td>
<td>NLS_DUAL_CURRENCY</td>
</tr>
<tr>
<td></td>
<td>NLS_ISO_CURRENCY</td>
</tr>
<tr>
<td>TO_NCHAR</td>
<td>NLS_DATE_LANGUAGE</td>
</tr>
<tr>
<td></td>
<td>NLS_NUMERIC_CHARACTERS</td>
</tr>
<tr>
<td></td>
<td>NLSCurrency</td>
</tr>
<tr>
<td></td>
<td>NLS_DUAL_CURRENCY</td>
</tr>
<tr>
<td></td>
<td>NLS_ISO_CURRENCY</td>
</tr>
<tr>
<td></td>
<td>NLSCALENDAR</td>
</tr>
<tr>
<td>NLS_UPPER</td>
<td>NLS_SORT</td>
</tr>
<tr>
<td>NLSLOWER</td>
<td>NLS_SORT</td>
</tr>
<tr>
<td>NLSINITCAP</td>
<td>NLS_SORT</td>
</tr>
<tr>
<td>NLS_SORT</td>
<td>NLS_SORT</td>
</tr>
</tbody>
</table>

The following examples show how to use NLS parameters in SQL functions:

```sql
TO_DATE ('1-JAN-99', 'DD-MON-YY',
          'nls_date_language = American')
```

```sql
TO_CHAR (hire_date, 'DD/MON/YYYY',
          'nls_date_language = French')
```

```sql
TO_CHAR (SYSDATE, 'DD/MON/YYYY',
          'nls_date_language=''Traditional Chinese''
```

```sql
TO_NUMBER ('13.000,00', '99G999D99',
            'nls_numeric_characters =','',')
```

```sql
TO_CHAR (salary, '9G999D99L', 'nls_numeric_characters = '''
          nls_currency = 'EUR'')
```

```sql
TO_CHAR (salary, '9G999D99C', 'nls_numeric_characters = '''
```
nls_iso_currency = Japan')
NLS_UPPER (last_name, 'nls_sort = Swiss')
NLSSORT (last_name, 'nls_sort = German')

**Note:** In some languages, some lowercase characters correspond to more than one uppercase character or vice versa. As a result, the length of the output from the NLS_UPPER, NLS_LOWER, and NLS_INITCAP functions can differ from the length of the input.

See Also: “Special Uppercase Letters” on page 5-11 and “Special Lowercase Letters” on page 5-11

### Unacceptable NLS Parameters in SQL Functions

The following NLS parameters are not accepted in SQL functions except for NLSSORT:

- NLS_LANGUAGE
- NLS_TERRITORY
- NLS_DATE_FORMAT

NLS_DATE_FORMAT and NLS_TERRITORY_FORMAT are not accepted as parameters because they can interfere with required format masks. A date format must always be specified if an NLS parameter is in a TO_CHAR or TO_DATE function. As a result, NLS_DATE_FORMAT and NLS_TERRITORY_FORMAT are not valid NLS parameters for the TO_CHAR or TO_DATE functions. If you specify NLS_DATE_FORMAT or NLS_TERRITORY_FORMAT in the TO_CHAR or TO_DATE function, then an error is returned.

NLS_LANGUAGE can interfere with the session value of NLS_DATE_LANGUAGE. If you specify NLS_LANGUAGE in the TO_CHAR function, for example, then its value is ignored if it differs from the session value of NLS_DATE_LANGUAGE.

### Other Locale-Dependent SQL Functions

This section includes the following topics:

- The CONVERT Function
The CONVERT Function

The CONVERT function enables conversion of character data between character sets.

The CONVERT function converts the binary representation of a character string in one character set to another. It uses exactly the same technique as conversion between database and client character sets. Hence, it uses replacement characters and has the same limitations.

See Also: "Character Set Conversion Between Clients and the Server" on page 2-17

The syntax for CONVERT is as follows:

CONVERT(char, dest_char_set[, source_char_set])

char is the value to be converted. source_char_set is the source character set and dest_char_set is the destination character set. If the source_char_set parameter is not specified, then it defaults to the database character set.

See Also:

- Oracle Database SQL Reference for more information about the CONVERT function
- "Character Set Conversion Support" on page A-19 for character set encodings that are used only for the CONVERT function

SQL Functions for Different Length Semantics

Oracle provides SQL functions that work in accordance with different length semantics. There are three groups of such SQL functions: SUBSTR, LENGTH, and INSTR. Each function in a group is based on a different kind of length semantics and is distinguished by the character or number appended to the function name. For example, SUBSTRB is based on byte semantics.
The SUBSTR functions return a requested portion of a substring. The LENGTH functions return the length of a string. The INSTR functions search for a substring in a string.

The SUBSTR functions calculate the length of a string differently. Table 9-1 summarizes the calculation methods.

### Table 9-2 How the SUBSTR Functions Calculate the Length of a String

<table>
<thead>
<tr>
<th>Function</th>
<th>Calculation Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBSTR</td>
<td>Calculates the length of a string in characters based on the length semantics associated with the character set of the datatype. For example, AL32UTF8 characters are calculated in UCS-4 characters. UTF8 and AL16UTF16 characters are calculated in UCS-2 characters. A supplementary character is counted as one character in AL32UTF8 and as two characters in UTF8 and AL16UTF16. Because VARCHAR and NVARCHAR may use different character sets, SUBSTR may give different results for different datatypes even if two strings are identical. If your application requires consistency, then use SUBSTR2 or SUBSTR4 to force all semantic calculations to be UCS-2 or UCS-4, respectively.</td>
</tr>
<tr>
<td>SUBSTRB</td>
<td>Calculates the length of a string in bytes</td>
</tr>
<tr>
<td>SUBSTR2</td>
<td>Calculates the length of a string in UCS-2 characters, which is compliant with Java strings and Windows client environments. Characters are represented in UCS-2 or 16-bit Unicode values. Supplementary characters are counted as two characters.</td>
</tr>
<tr>
<td>SUBSTR4</td>
<td>Calculates the length of a string in UCS-4 characters. Characters are represented in UCS-4 or 32-bit Unicode values. Supplementary characters are counted as one character.</td>
</tr>
<tr>
<td>SUBSTRC</td>
<td>Calculates the length of a string in Unicode composed characters. Supplementary characters and composed characters are counted as one character.</td>
</tr>
</tbody>
</table>

The LENGTH and INSTR functions calculate string length in the same way, according to the character or number added to the function name.

The following examples demonstrate the differences between SUBSTR and SUBSTRB on a database whose character set is AL32UTF8.

For the string **Fußball**, the following statement returns a substring that is 4 characters long, beginning with the second character:

```
SELECT SUBSTR ('Fußball', 2, 4) SUBSTR FROM dual;
```

<table>
<thead>
<tr>
<th>SUBS</th>
<th>----</th>
</tr>
</thead>
<tbody>
<tr>
<td>ußba</td>
<td></td>
</tr>
</tbody>
</table>

For the string **Fußball**, the following statement returns a substring 4 bytes long, beginning with the second byte:

```
SELECT SUBSTRB ('Fußball', 2, 4) SUBSTRB FROM dual;
```
SELECT SUBSTRB ('Fußball', 2, 4) SUBSTRB FROM dual;

See Also: Oracle Database SQL Reference for more information about the SUBSTR, LENGTH, and INSTR functions

LIKE Conditions for Different Length Semantics

The LIKE conditions specify a test that uses pattern-matching. The equality operator (=) exactly matches one character value to another, but the LIKE conditions match a portion of one character value to another by searching the first value for the pattern specified by the second.

LIKE calculates the length of strings in characters using the length semantics associated with the input character set. The LIKE2, LIKE4, and LIKEC conditions are summarized in Table 9–3.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIKE2</td>
<td>Use when characters are represented in UCS-2 semantics. A supplementary character is considered as two characters.</td>
</tr>
<tr>
<td>LIKE4</td>
<td>Use when characters are represented in UCS-4 semantics. A supplementary character is considered as one character.</td>
</tr>
<tr>
<td>LIKEC</td>
<td>Use when characters are represented in Unicode complete character semantics. A composed character is treated as one character.</td>
</tr>
</tbody>
</table>

There is no LIKEB condition.

Character Set SQL Functions

Two SQL functions, NLS_CHARSET_NAME and NLS_CHARSET_ID, can convert between character set ID numbers and character set names. They are used by programs that need to determine character set ID numbers for binding variables through OCI.

Another SQL function, NLS_CHARSETDECLLEN, returns the length of an NCHAR column.

This section includes the following topics:
- Converting from Character Set Number to Character Set Name
- Converting from Character Set Name to Character Set Number
- Returning the Length of an NCHAR Column

See Also: Oracle Database SQL Reference

Converting from Character Set Number to Character Set Name

The NLS_CHARSET_NAME(n) function returns the name of the character set corresponding to ID number n. The function returns NULL if n is not a recognized character set ID value.

Converting from Character Set Name to Character Set Number

NLS_CHARSET_ID(text) returns the character set ID corresponding to the name specified by text. text is defined as a run-time VARCHAR2 quantity, a character set name. Values for text can be NLSRTL names that resolve to character sets that are not the database character set or the national character set.

If the value CHAR_CS is entered for text, then the function returns the ID of the server's database character set. If the value NCHAR_CS is entered for text, then the function returns the ID of the server's national character set. The function returns NULL if text is not a recognized name.

Note: The value for text must be entered in uppercase characters.

Returning the Length of an NCHAR Column

NLS_CHARSET_DECL_LEN(BYTECNT, CSID) returns the declaration length of a column in number of characters. BYTECNT is the byte length of the column. CSID is the character set ID of the column.

The NLSSORT Function

The NLSSORT function enables you to use any linguistic sort for an ORDER BY clause. It replaces a character string with the equivalent sort string used by the linguistic sort mechanism so that sorting the replacement strings produces the desired sorting sequence. For a binary sort, the sort string is the same as the input string.
The kind of linguistic sort used by an ORDER BY clause is determined by the NLS_SORT session parameter, but it can be overridden by explicitly using the NLSSORT function.

Example 9–1 specifies a German sort with the NLS_SORT session parameter.

**Example 9–1 Specifying a German Sort with the NLS_SORT Session Parameter**

ALTER SESSION SET NLS_SORT = GERMAN;
SELECT * FROM table1
    ORDER BY column1;

**Example 9–2 Specifying a French Sort with the NLSSORT Function**

This example first sets the NLS_SORT session parameter to German, but the NLSSORT function overrides it by specifying a French sort.

ALTER SESSION SET NLS_SORT = GERMAN;
SELECT * FROM table1
    ORDER BY NLSSORT(column1, 'NLS_SORT=FRENCH');

The WHERE clause uses binary comparison rather than linguistic comparison by default, but this can be overridden by using the NLSSORT function in the WHERE clause.

**Example 9–3 Making a Linguistic Comparison with the WHERE Clause**

ALTER SESSION SET NLS_COMP = ANSI;
SELECT * FROM table1
WHERE NLSSORT(column1, 'NLS_SORT=FRENCH')>
    NLSSORT(column2, 'NLS_SORT=FRENCH');

Setting the NLS_COMP session parameter to ANSI causes the NLS_SORT value to be used in the WHERE clause.

The rest of this section contains the following topics:

- NLSSORT Syntax
- Comparing Strings in a WHERE Clause
- Using the NLS_COMP Parameter to Simplify Comparisons in the WHERE Clause
- Controlling an ORDER BY Clause
**NLSSORT Syntax**

There are four ways to use NLSSORT:

- NLSSORT(), which relies on the NLS_SORT parameter
- NLSSORT(column1, 'NLS_SORT=xxxx')
- NLSSORT(column1, 'NLS_LANG=xxxx')
- NLSSORT(column1, 'NLS_LANGUAGE=xxxx')

The NLS_LANG parameter of the NLSSORT function is not the same as the NLS_LANG client environment setting. In the NLSSORT function, NLS_LANG specifies the abbreviated language name, such as US for American or PL for Polish. For example:

```sql
SELECT * FROM table1
ORDER BY NLSSORT(column1, 'NLS_LANG=PL');
```

**Comparing Strings in a WHERE Clause**

NLSSORT enables applications to perform string matching that follows alphabetic conventions. Normally, character strings in a WHERE clause are compared by using the binary values of the characters. One character is considered greater than another character if it has a greater binary value in the database character set. Because the sequence of characters based on their binary values might not match the alphabetic sequence for a language, such comparisons may not follow alphabetic conventions.

For example, if a column (column1) contains the values ABC, ABZ, BCD, and ÄBC in the ISO 8859-1 8-bit character set, then the following query returns both BCD and ÄBC because Ä has a higher numeric value than B:

```sql
SELECT column1 FROM table1 WHERE column1 > 'B';
```

In German, Ä is sorted alphabetically before B, but in Swedish, Ä is sorted after Z. Linguistic comparisons can be made by using NLSSORT in the WHERE clause:

```sql
WHERE NLSSORT(col) comparison_operator NLSSORT(comparison_string)
```

Note that NLSSORT must be on both sides of the comparison operator. For example:

```sql
SELECT column1 FROM table1 WHERE NLSSORT(column1) > NLSSORT('B');
```

If a German linguistic sort has been set, then the statement does not return strings beginning with Ä because Ä comes before B in the German alphabet. If a Swedish linguistic sort has been set, then strings beginning with Ä are returned because Ä comes after Z in the Swedish alphabet.
Using the NLS_COMP Parameter to Simplify Comparisons in the WHERE Clause

Comparison in the WHERE clause or PL/SQL blocks is binary by default. Using the NLSSORT function for linguistic comparison can be tedious, especially when the linguistic sort has already been specified in the NLS_SORT session parameter. You can use the NLS_COMP parameter to indicate that the comparisons in a WHERE clause or in PL/SQL blocks must be linguistic according to the NLS_SORT session parameter.

**Note:** The NLS_COMP parameter does not affect comparison behavior for partitioned tables. String comparisons that are based on a VALUES LESS THAN partition are always binary.

See Also: "NLS_COMP" on page 3-42

Controlling an ORDER BY Clause

If a linguistic sort is in use, then ORDER BY clauses use an implicit NLSSORT on character data. The sort mechanism (linguistic or binary) for an ORDER BY clause is transparent to the application. However, if the NLSSORT function is explicitly specified in an ORDER BY clause, then the implicit NLSSORT is not done.

If a linguistic sort has been defined by the NLS_SORT session parameter, then an ORDER BY clause in an application uses an implicit NLSSORT function. If you specify an explicit NLSSORT function, then it overrides the implicit NLSSORT function.

When the sort mechanism has been defined as linguistic, the NLSSORT function is usually unnecessary in an ORDER BY clause.

When the sort mechanism either defaults or is defined as binary, then a query like the following uses a binary sort:

```sql
SELECT last_name FROM employees
ORDER BY last_name;
```

A German linguistic sort can be obtained as follows:

```sql
SELECT last_name FROM employees
ORDER BY NLSSORT(last_name, 'NLS_SORT = GERMAN');
```

See Also: "Using Linguistic Sorts" on page 5-3
Miscellaneous Topics for SQL and PL/SQL Programming in a Global Environment

This section contains the following topics:

- SQL Date Format Masks
- Calculating Week Numbers
- SQL Numeric Format Masks
- Loading External BFILE Data into LOB Columns

See Also: Oracle Database SQL Reference for a complete description of format masks

SQL Date Format Masks

Several format masks are provided with the `TO_CHAR`, `TO_DATE`, and `TO_NUMBER` functions.

The `RM` (Roman Month) format element returns a month as a Roman numeral. You can specify either upper case or lower case by using `RM` or `rm`. For example, for the date 7 Sep 1998, `DD-rm-YYYY` returns `07-ix-1998` and `DD-RM-YYYY` returns `07-IX-1998`.

Note that the `MON` and `DY` format masks explicitly support month and day abbreviations that may not be three characters in length. For example, the abbreviations "Lu" and "Ma" can be specified for the French "Lundi" and "Mardi", respectively.

Calculating Week Numbers

The week numbers returned by the `WW` format mask are calculated according to the following algorithm: \( \text{int}(\text{dayOfYear} + 6)/7 \). This algorithm does not follow the ISO standard (2015, 1992-06-15).

To support the ISO standard, the `IW` format element is provided. It returns the ISO week number. In addition, the `I`, `IY`, `IYY`, and `YYYY` format elements, equivalent in behavior to the `Y`, `YY`, `YYY`, and `YYYY` format elements, return the year relating to the ISO week number.

In the ISO standard, the year relating to an ISO week number can be different from the calendar year. For example, 1st Jan 1988 is in ISO week number 53 of 1987.
week always starts on a Monday and ends on a Sunday. The week number is
determined according the following rules:

- If January 1 falls on a Friday, Saturday, or Sunday, then the week including
  January 1 is the last week of the previous year, because most of the days in the
  week belong to the previous year.
- If January 1 falls on a Monday, Tuesday, Wednesday, or Thursday, then the
  week is the first week of the new year, because most of the days in the week
  belong to the new year.

For example, January 1, 1991, is a Tuesday, so Monday, December 31, 1990, to
Sunday, January 6, 1991, is in week 1. Thus, the ISO week number and year for
December 31, 1990, is 1, 1991. To get the ISO week number, use the \_w format mask
for the week number and one of the \_Y formats for the year.

### SQL Numeric Format Masks

Several additional format elements are provided for formatting numbers:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Decimal</td>
<td>Returns the decimal point character</td>
</tr>
<tr>
<td>G</td>
<td>Group</td>
<td>Returns the group separator</td>
</tr>
<tr>
<td>L</td>
<td>Local currency</td>
<td>Returns the local currency symbol</td>
</tr>
<tr>
<td>C</td>
<td>International currency</td>
<td>Returns the ISO currency symbol</td>
</tr>
<tr>
<td>RN</td>
<td>Roman numeral</td>
<td>Returns the number as its Roman numeral equivalent</td>
</tr>
</tbody>
</table>

For Roman numerals, you can specify either upper case or lower case, using RN or rn, respectively. The number being converted must be an integer in the range 1 to 3999.

### Loading External BFILE Data into LOB Columns

The DBMS_LOB PL/SQL package can load external BFILE data into LOB columns. Previous releases of Oracle did not perform character set conversion before loading the binary data into CLOB or NCLOB columns. Thus the BFILE data had to be in the same character set as the database or national character set to work properly. The APIs that were introduced in Oracle9i Release 2 (9.2) allow the user to specify the character set ID of the BFILE data by using a new parameter. The APIs convert the data from the specified BFILE character set into the database character set for the
CLOB datatype or the national character set for the NCLOB datatype. The loading takes place on the server because BFILE data is not supported on the client.

- Use `DBMS_LOB.LOADBLOBFROMFILE` to load BLOB columns.
- Use `DBMS_LOB.LOADCLOBFROMFILE` to load CLOB and NCLOB columns.

See Also:
- PL/SQL Packages and Types Reference
- Oracle Database Application Developer’s Guide - Large Objects
This chapter contains information about OCI programming in a global environment. It includes the following topics:

- Using the OCI NLS Functions
- Specifying Character Sets in OCI
- Getting Locale Information in OCI
- Mapping Locale Information Between Oracle and Other Standards
- Manipulating Strings in OCI
- Classifying Characters in OCI
- Converting Character Sets in OCI
- OCI Messaging Functions
- lmsgen Utility
Using the OCI NLS Functions

Many OCI NLS functions accept one of the following handles:

- The environment handle
- The user session handle

The OCI environment handle is associated with the client NLS environment and initialized with the client NLS environment variables. This environment does not change when ALTER SESSION statements are issued to the server. The character set associated with the environment handle is the client character set.

The OCI session handle is associated with the server session environment. Its NLS settings change when the session environment is modified with an ALTER SESSION statement. The character set associated with the session handle is the database character set.

Note that the OCI session handle does not have any NLS settings associated with it until the first transaction begins in the session. SELECT statements do not begin a transaction.

See Also: Oracle Call Interface Programmer’s Guide for detailed information about the OCI NLS functions

Specifying Character Sets in OCI

Use the OCIEnvNlsCreate function to specify client-side database and national character sets when the OCI environment is created. This function allows users to set character set information dynamically in applications, independent of the NLS_LANG and NLS_NCHAR initialize parameter settings. In addition, one application can initialize several environment handles for different client environments in the same server environment.

Any Oracle character set ID except AL16UTF16 can be specified through the OCIEnvNlsCreate function to specify the encoding of metadata, SQL CHAR data, and SQL NCHAR data. Use OCI_UTF16ID in the OCIEnvNlsCreate function to specify UTF-16 data.

See Also: Oracle Call Interface Programmer’s Guide for more information about the OCIEnvNlsCreate function
Getting Locale Information in OCI

An Oracle locale consists of language, territory, and character set definitions. The locale determines conventions such as day and month names, as well as date, time, number, and currency formats. A globalized application complies with a user’s locale setting and cultural conventions. For example, when the locale is set to German, users expect to see day and month names in German.

You can use the `OCINlsGetInfo()` function to retrieve the following locale information:

- Days of the week (translated)
- Abbreviated days of the week (translated)
- Month names (translated)
- Abbreviated month names (translated)
- Yes/no (translated)
- AM/PM (translated)
- AD/BC (translated)
- Numeric format
- Debit/credit
- Date format
- Currency formats
- Default language
- Default territory
- Default character set
- Default linguistic sort
- Default calendar

Table 10–1 summarizes OCI functions that return locale information.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OCINlsGetInfo()</code></td>
<td>Returns locale information. See preceding text.</td>
</tr>
<tr>
<td><code>OCINlsCharSetNameTold()</code></td>
<td>Returns the Oracle character set ID for the specified Oracle character set name</td>
</tr>
<tr>
<td><code>OCINlsCharSetIdToName()</code></td>
<td>Returns the Oracle character set name from the specified character set ID</td>
</tr>
<tr>
<td><code>OCINlsNumericInfoGet()</code></td>
<td>Returns specified numeric information such as maximum character size</td>
</tr>
<tr>
<td><code>OCINlsEnvironmentVariableGet()</code></td>
<td>Returns the character set ID from <code>NLS_LANG</code> or the national character set ID from <code>NLS_NCHAR</code></td>
</tr>
</tbody>
</table>
Mapping Locale Information Between Oracle and Other Standards

The OCINlsNameMap function maps Oracle character set names, language names, and territory names to and from Internet Assigned Numbers Authority (IANA) and International Organization for Standardization (ISO) names.

Manipulating Strings in OCI

Two types of data structures are supported for string manipulation:

- Native character strings
- Wide character strings

Native character strings are encoded in native Oracle character sets. Functions that operate on native character strings take the string as a whole unit with the length of the string calculated in bytes. Wide character (wchar) string functions provide more flexibility in string manipulation. They support character-based and string-based operations with the length of the string calculated in characters.

The wide character datatype is Oracle-specific and should not be confused with the wchar_t datatype defined by the ANSI/ISO C standard. The Oracle wide character datatype is always 4 bytes in all platforms, while the size of wchar_t depends on the implementation and the platform. The Oracle wide character datatype normalizes native characters so that they have a fixed width for easy processing. This guarantees no data loss for round-trip conversion between the Oracle wide character format and the native character format.

String manipulation can be classified as followings:

- Conversion of strings between native character format and wide character format
- Character classifications
- Case conversion
- Calculations of display length
- General string manipulation, such as comparison, concatenation, and searching

Table 10–2 summarizes the OCI string manipulation functions.
Note: The functions and descriptions in Table 10-2 that refer to multibyte strings apply to native character strings.

**Table 10–2  OCI String Manipulation Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCIMultiByteToWideChar()</td>
<td>Converts an entire null-terminated string into the wchar format</td>
</tr>
<tr>
<td>OCIMultiByteInSizeToWideChar()</td>
<td>Converts part of a string into the wchar format</td>
</tr>
<tr>
<td>OCIWideCharToMultiByte()</td>
<td>Converts an entire null-terminated wide character string into a multibyte string</td>
</tr>
<tr>
<td>OCIWideCharInSizeToMultiByte()</td>
<td>Converts part of a wide character string into the multibyte format</td>
</tr>
<tr>
<td>OCIWideCharToLower()</td>
<td>Converts the wchar character specified by wc into the corresponding lowercase character if it exists in the specified locale. If no corresponding lowercase character exists, then it returns wc itself.</td>
</tr>
<tr>
<td>OCIWideCharToUpper()</td>
<td>Converts the wchar character specified by wc into the corresponding uppercase character if it exists in the specified locale. If no corresponding uppercase character exists, then it returns wc itself.</td>
</tr>
<tr>
<td>OCIWideCharStrcmp()</td>
<td>Compares two wide character strings by binary, linguistic, or case-insensitive comparison method</td>
</tr>
<tr>
<td>OCIWideCharStrncmp()</td>
<td>Similar to OCIWideCharStrcmp(). Compares two wide character strings by binary, linguistic, or case-insensitive comparison methods. At most len1 bytes form str1, and len2 bytes form str2.</td>
</tr>
<tr>
<td>OCIWideCharStrcat()</td>
<td>Appends a copy of the string pointed to by wsrcstr. Then it returns the number of characters in the resulting string.</td>
</tr>
<tr>
<td>OCIWideCharStrncat()</td>
<td>Appends a copy of the string pointed to by wsrcstr. Then it returns the number of characters in the resulting string. At most n characters are appended.</td>
</tr>
<tr>
<td>OCIWideCharStrchr()</td>
<td>Searches for the first occurrence of wc in the string pointed to by wstr. Then it returns a pointer to the wchar if the search is successful.</td>
</tr>
<tr>
<td>OCIWideCharStrrchr()</td>
<td>Searches for the last occurrence of wc in the string pointed to by wstr</td>
</tr>
<tr>
<td>OCIWideCharStrncpy()</td>
<td>Copies the wchar string pointed to by wsrcstr into the array pointed to by wdststr. Then it returns the number of characters copied.</td>
</tr>
</tbody>
</table>
Table 10–2  OCI String Manipulation Functions (Cont.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCIWideCharStrncpy()</td>
<td>Copies the wchar string pointed to by wsrcstr into the array pointed to by wdststr. Then it returns the number of characters copied. At most $n$ characters are copied from the array.</td>
</tr>
<tr>
<td>OCIWideCharStrlen()</td>
<td>Computes the number of characters in the wchar string pointed to by wstr and returns this number.</td>
</tr>
<tr>
<td>OCIWideCharStrCaseConversion()</td>
<td>Converts the wide character string pointed to by wsrcstr into the case specified by a flag and copies the result into the array pointed to by wdststr.</td>
</tr>
<tr>
<td>OCIWideCharDisplayLength()</td>
<td>Determines the number of column positions required for wc in display.</td>
</tr>
<tr>
<td>OCIWideCharMultibyteLength()</td>
<td>Determines the number of bytes required for wc in multibyte encoding.</td>
</tr>
<tr>
<td>OCIMultiByteStrcmp()</td>
<td>Compares two multibyte strings by binary, linguistic, or case-insensitive comparison methods.</td>
</tr>
<tr>
<td>OCIMultiByteStrncmp()</td>
<td>Compares two multibyte strings by binary, linguistic, or case-insensitive comparison methods. At most $\text{len}_1$ bytes form $\text{str}_1$ and $\text{len}_2$ bytes form $\text{str}_2$.</td>
</tr>
<tr>
<td>OCIMultiByteStrcat()</td>
<td>Appends a copy of the multibyte string pointed to by $\text{srcstr}$ to $\text{dststr}$.</td>
</tr>
<tr>
<td>OCIMultiByteStrncat()</td>
<td>Appends a copy of the multibyte string pointed to by $\text{srcstr}$. At most $n$ bytes from $\text{srcstr}$ are appended to $\text{dststr}$.</td>
</tr>
<tr>
<td>OCIMultiByteStrcpy()</td>
<td>Copies the multibyte string pointed to by $\text{srcstr}$ into an array pointed to by $\text{dststr}$. It returns the number of bytes copied.</td>
</tr>
<tr>
<td>OCIMultiByteStrncpy()</td>
<td>Copies the multibyte string pointed to by $\text{srcstr}$ into an array pointed to by $\text{dststr}$. It returns the number of bytes copied. At most $n$ bytes are copied from the array pointed to by $\text{srcstr}$ to the array pointed to by $\text{dststr}$.</td>
</tr>
<tr>
<td>OCIMultiByteStrlen()</td>
<td>Returns the number of bytes in the multibyte string pointed to by $\text{str}$.</td>
</tr>
<tr>
<td>OCIMultiByteStrnDisplayLength()</td>
<td>Returns the number of display positions occupied by the complete characters within the range of $n$ bytes.</td>
</tr>
<tr>
<td>OCIMultiByteStrCaseConversion()</td>
<td>Converts part of a string from one character set to another.</td>
</tr>
</tbody>
</table>

See Also:  *Oracle Call Interface Programmer’s Guide*
Classifying Characters in OCI

Table 10–3 shows the OCI character classification functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCIWideCharIsAlnum()</td>
<td>Tests whether the wide character is an alphabetic letter or decimal digit</td>
</tr>
<tr>
<td>OCIWideCharIsAlpha()</td>
<td>Tests whether the wide character is an alphabetic letter</td>
</tr>
<tr>
<td>OCIWideCharIsCntrl()</td>
<td>Tests whether the wide character is a control character</td>
</tr>
<tr>
<td>OCIWideCharIsDigit()</td>
<td>Tests whether the wide character is a decimal digit</td>
</tr>
<tr>
<td>OCIWideCharIsGraph()</td>
<td>Tests whether the wide character is a graph character</td>
</tr>
<tr>
<td>OCIWideCharIsLower()</td>
<td>Tests whether the wide character is a lowercase letter</td>
</tr>
<tr>
<td>OCIWideCharIsPrint()</td>
<td>Tests whether the wide character is a printable character</td>
</tr>
<tr>
<td>OCIWideCharIsPunct()</td>
<td>Tests whether the wide character is a punctuation character</td>
</tr>
<tr>
<td>OCIWideCharIsSpace()</td>
<td>Tests whether the wide character is a space character</td>
</tr>
<tr>
<td>OCIWideCharIsUpper()</td>
<td>Tests whether the wide character is an uppercase character</td>
</tr>
<tr>
<td>OCIWideCharIsXdigit()</td>
<td>Tests whether the wide character is a hexadecimal digit</td>
</tr>
<tr>
<td>OCIWideCharIsSingleByte()</td>
<td>Tests whether ( wc ) is a single-byte character when converted into multibyte</td>
</tr>
</tbody>
</table>

See Also: Oracle Call Interface Programmer’s Guide

Converting Character Sets in OCI

Conversion between Oracle character sets and Unicode (16-bit, fixed-width Unicode encoding) is supported. Replacement characters are used if a character has no mapping from Unicode to the Oracle character set. Therefore, conversion back to the original character set is not always possible without data loss.

Table 10–4 summarizes the OCI character set conversion functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCICharSetToUnicode()</td>
<td>Converts a multibyte string pointed to by ( src ) to Unicode into the array pointed to by ( dst )</td>
</tr>
<tr>
<td>OCIUnicodeToCharSet()</td>
<td>Converts a Unicode string pointed to by ( src ) to multibyte into the array pointed to by ( dst )</td>
</tr>
</tbody>
</table>

OCI Programming in a Global Environment 10-7
OCI Messaging Functions

Table 10–4 OCI Character Set Conversion Functions (Cont.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCINlsCharSetConvert()</td>
<td>Converts a string from one character set to another</td>
</tr>
<tr>
<td>OCICharSetConversionIsReplacementUsed()</td>
<td>Indicates whether replacement characters were used for characters that could not be converted in the last invocation of OCINlsCharSetConvert() or OCIUnicodeToCharSet()</td>
</tr>
</tbody>
</table>

See Also:
- Oracle Call Interface Programmer’s Guide
- "OCI Programming with Unicode" on page 7-13

OCI Messaging Functions

The user message API provides a simple interface for cartridge developers to retrieve their own messages as well as Oracle messages.

Table 10–5 summarizes the OCI messaging functions.

Table 10–5 OCI Messaging Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCIMessageOpen()</td>
<td>Opens a message handle in a language pointed to by hndl</td>
</tr>
<tr>
<td>OCIMessageGet()</td>
<td>Retrieves a message with message number identified by msgno. If the buffer is not zero, then the function copies the message into the buffer specified by msgbuf.</td>
</tr>
<tr>
<td>OCIMessageClose()</td>
<td>Closes a message handle pointed to by msgh and frees any memory associated with this handle</td>
</tr>
</tbody>
</table>

See Also: Oracle Data Cartridge Developer’s Guide

Imsgen Utility

Purpose

The Imsgen utility converts text-based message files (.msg) into binary format (.msb) so that Oracle messages and OCI messages provided by the user can be returned to OCI functions in the desired language.
Syntax

LMSGEN text_file product facility [language]

text_file is a message text file.
product is the name of the product.
facility is the name of the facility.

language is the optional message language corresponding to the language specified in the NLS_LANG parameter. The language parameter is required if the message file is not tagged properly with language.

Text Message Files

Text message files must follow these guidelines:

- Lines that start with / and // are treated as internal comments and are ignored.
- To tag the message file with a specific language, include a line similar to the following:
  
  # CHARACTER_SET_NAME= Japanese_Japan.JA16EUC

- Each message contains 3 fields:
  
  message_number, warning_level, message_text

  The message number must be unique within a message file.
  The warning level is not currently used. Use 0.
  The message text cannot be longer than 511 bytes.

The following is an example of an Oracle message text file:

/ Copyright (c) 2001 by the Oracle Corporation. All rights reserved.
/ This is a test us7ascii message file
# CHARACTER_SET_NAME= american_america.us7ascii
/
00000, 00000, "Export terminated unsuccessfully\n"
00003, 00000, "no storage definition found for segment(\%lu, \%lu)"

Example: Creating a Binary Message File from a Text Message File

The following table contains sample values for the lmsgen parameters:
The text message file is found in the following location:

$HOME/myApp/mesg/impus.msg

One of the lines in the text message file is:

00128,2, "Duplicate entry %s found in %s"

The lmsgen utility converts the text message file (impus.msg) into binary format, resulting in a file called impus.msb:

% lmsgen impus.msg $HOME/myApplication imp AMERICAN

The following output results:

Generating message file impus.msg -->
/home/scott/myApplication/mesg/impus.msb

NLS Binary Message File Generation Utility: Version 10.1.0.0.0 -Production

Copyright (c) Oracle Corporation 1979, 2003. All rights reserved.

CORE 10.1.0.0.0 Production
This chapter discusses character set conversion and character set migration. It includes the following topics:

- Overview of Character Set Migration
- Changing the Database Character Set of an Existing Database
- Migrating to NCHAR Datatypes
- Tasks to Recover Database Schema After Character Set Migration
Overview of Character Set Migration

Choosing the appropriate character set for your database is an important decision. When you choose the database character set, consider the following factors:

- The type of data you need to store
- The languages that the database needs to accommodate now and in the future
- The different size requirements of each character set and the corresponding performance implications

A related topic is choosing a new character set for an existing database. Changing the database character set for an existing database is called character set migration. When you migrate from one database character set to another you must choose an appropriate character set. You should also plan to minimize data loss from the following sources:

- Data Truncation
- Character Set Conversion Issues

See Also: Chapter 2, "Choosing a Character Set"

Data Truncation

When the database is created using byte semantics, the sizes of the CHAR and VARCHAR2 datatypes are specified in bytes, not characters. For example, the specification CHAR(20) in a table definition allows 20 bytes for storing character data. When the database character set uses a single-byte character encoding scheme, no data loss occurs when characters are stored because the number of characters is equivalent to the number of bytes. If the database character set uses a multibyte character set, then the number of bytes no longer equals the number of characters because a character can consist of one or more bytes.

During migration to a new character set, it is important to verify the column widths of existing CHAR and VARCHAR2 columns because they may need to be extended to support an encoding that requires multibyte storage. Truncation of data can occur if conversion causes expansion of data.

Table 11–1 shows an example of data expansion when single-byte characters become multibyte characters through conversion.
The first column of Table 11–1 shows selected characters. The second column shows the hexadecimal representation of the characters in the WE8MSWIN1252 character set. The third column shows the hexadecimal representation of each character in the AL32UTF8 character set. Each pair of letters and numbers represents one byte. For example, ä (a with an umlaut) is a single-byte character (E4) in WE8MSWIN1252, but it becomes a two-byte character (C3 A4) in AL32UTF8. Also, the encoding for the euro symbol expands from one byte (80) to three bytes (E2 82 AC).

If the data in the new character set requires storage that is greater than the supported byte size of the datatypes, then you need to change your schema. You may need to use CLOB columns.

See Also: "Length Semantics" on page 2-12

### Additional Problems Caused by Data Truncation

Data truncation can cause the following problems:

- In the database data dictionary, schema object names cannot exceed 30 bytes in length. You must rename schema objects if their names exceed 30 bytes in the new database character set. For example, one Thai character in the Thai national character set requires 1 byte. In AL32UTF8, it requires 3 bytes. If you have defined a table whose name is 11 Thai characters, then the table name must be shortened to 10 or fewer Thai characters when you change the database character set to AL32UTF8.

- If existing Oracle usernames or passwords are created based on characters that change in size in the new character set, then users will have trouble logging in because of authentication failures after the migration to a new character set. This occurs because the encrypted usernames and passwords stored in the data dictionary may not be updated during migration to a new character set. For example, if the current database character set is WE8MSWIN1252 and the new database character set is AL32UTF8, then the length of the username scött (ö...
with an umlaut) changes from 5 bytes to 6 bytes. In AL32UTF8, \texttt{scott} can no longer log in because of the difference in the username. Oracle Corporation recommends that usernames and passwords be based on ASCII characters. If they are not, then you must reset the affected usernames and passwords after migrating to a new character set.

\textbf{Note:} Encrypted usernames and passwords stored in the data dictionary are not updated when migration is accomplished with the \texttt{ALTER DATABASE} method, but they are updated if the migration is accomplished with the Import and Export utilities.

- When \texttt{CHAR} data contains characters that expand after migration to a new character set, space padding is not removed during database export by default. This means that these rows will be rejected upon import into the database with the new character set. The workaround is to set the \texttt{BLANK_TRIMMING} initialization parameter to \texttt{TRUE} before importing the \texttt{CHAR} data.

\textbf{See Also:} \textit{Oracle Database Reference} for more information about the \texttt{BLANK_TRIMMING} initialization parameter.

### Character Set Conversion Issues

This section includes the following topics:

- Replacement Characters that Result from Using the Export and Import Utilities
- Invalid Data That Results from Setting the Client’s NLS_LANG Parameter Incorrectly

#### Replacement Characters that Result from Using the Export and Import Utilities

The Export and Import utilities can convert character sets from the original database character set to the new database character set. However, character set conversions can sometimes cause data loss or data corruption. For example, if you are migrating from character set A to character set B, then the destination character set B should be a superset of character set A. The destination character, B, is a \texttt{superset} if it contains all the characters defined in character set A. Characters that are not available in character set B are converted to replacement characters, which are often specified as ? or \texttt{¿} or as a character that is related to the unavailable character. For example, \texttt{ä} (a with an umlaut) can be replaced by \texttt{a}. Replacement characters are defined by the target character set.
Overview of Character Set Migration

**Note:** There is an exception to the requirement that the destination character set B should be a superset of character set A. If your data contains no characters that are in character set A but are not in character set B, then the destination character set does not need to be a superset of character set A to avoid data loss or data corruption.

Figure 11–1 shows an example of a character set conversion in which the copyright and euro symbols are converted to ? and ä is converted to a.

**Figure 11–1 Replacement Characters in Character Set Conversion**

<table>
<thead>
<tr>
<th>Character Set A</th>
<th>Character Set B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>€</td>
<td>?</td>
</tr>
<tr>
<td>ä</td>
<td></td>
</tr>
</tbody>
</table>

To reduce the risk of losing data, choose a destination character set with a similar character repertoire. Migrating to Unicode can be an attractive option because AL32UTF8 contains characters from most legacy character sets.

**Invalid Data That Results from Setting the Client’s NLS_LANG Parameter Incorrectly**

Another character set migration scenario that can cause the loss of data is migrating a database that contains invalid data. Invalid data usually occurs in a database because the NLS_LANG parameter is not set properly on the client. The NLS_LANG value should reflect the client operating system code page. For example, in an English Windows environment, the code page is WE8MSWIN1252. When the NLS_LANG parameter is set properly, the database can automatically convert incoming data from the client operating system. When the NLS_LANG parameter is not set properly, then the data coming into the database is not converted properly. For
example, suppose that the database character set is AL32UTF8, the client is an English Windows operating system, and the NLS_LANG setting on the client is AL32UTF8. Data coming into the database is encoded in WE8MSWIN1252 and is not converted to AL32UTF8 data because the NLS_LANG setting on the client matches the database character set. Thus Oracle assumes that no conversion is necessary, and invalid data is entered into the database.

This can lead to two possible data inconsistency problems. One problem occurs when a database contains data from a character set that is different from the database character set but the same code points exist in both character sets. For example, if the database character set is WE8ISO8859P1 and the NLS_LANG setting of the Chinese Windows NT client is SIMPLIFIED CHINESE\_CHINA.WE8ISO8859P1, then all multibyte Chinese data (from the ZHS16GBK character set) is stored as multiples of single-byte WE8ISO8859P1 data. This means that Oracle treats these characters as single-byte WE8ISO8859P1 characters. Hence all SQL string manipulation functions such as SUBSTR or LENGTH are based on bytes rather than characters. All bytes constituting ZHS16GBK data are legal WE8ISO8859P1 codes. If such a database is migrated to another character set such as AL32UTF8, then character codes are converted as if they were in WE8ISO8859P1. This way, each of the two bytes of a ZHS16GBK character are converted separately, yielding meaningless values in AL32UTF8. Figure 11–2 shows an example of this incorrect character set replacement.

**Figure 11–2 Incorrect Character Set Replacement**

![Diagram of incorrect character set replacement](image)

The second possible problem is having data from mixed character sets inside the database. For example, if the data character set is WE8MSWIN1252, and two separate Windows clients using German and Greek are both using WE8MSWIN1252 as the NLS_LANG character set, then the database contains a
mixture of German and Greek characters. Figure 11–3 shows how different clients can use different character sets in the same database.

Figure 11–3 Mixed Character Sets

For database character set migration to be successful, both of these cases require manual intervention because Oracle cannot determine the character sets of the data being stored. Incorrect data conversion can lead to data corruption, so perform a full backup of the database before attempting to migrate the data to a new character set.

Changing the Database Character Set of an Existing Database

Database character set migration has two stages: data scanning and data conversion. Before you change the database character set, you need to identify possible database character set conversion problems and truncation of data. This step is called data scanning.

Data scanning identifies the amount of effort required to migrate data into the new character encoding scheme before changing the database character set. Some
examples of what may be found during a data scan are the number of schema objects where the column widths need to be expanded and the extent of the data that does not exist in the target character repertoire. This information helps to determine the best approach for converting the database character set.

Incorrect data conversion can lead to data corruption, so perform a full backup of the database before attempting to migrate the data to a new character set.

There are three approaches to converting data from one database character set to another if the database does not contain any of the inconsistencies described in "Character Set Conversion Issues" on page 11-4. A description of methods to migrate databases with such inconsistencies is out of the scope of this documentation. For more information, contact Oracle Consulting Services for assistance.

The approaches are:

- Migrating Character Data Using a Full Export and Import
- Migrating a Character Set Using the CSALTER Script
- Migrating Character Data Using the CSALTER Script and Selective Imports

**See Also:** Chapter 12, "Character Set Scanner Utilities" for more information about data scanning

### Migrating Character Data Using a Full Export and Import

In most cases, a full export and import is recommended to properly convert all data to a new character set. It is important to be aware of data truncation issues, because columns with character datatypess may need to be extended before the import to handle an increase in size. Existing PL/SQL code should be reviewed to ensure that all byte-based SQL functions such as `LENGTHB`, `SUBSTRB`, and `INSTRB`, and PL/SQL `CHAR` and `VARCHAR2` declarations are still valid.

**See Also:** Oracle Database Utilities for more information about the Export and Import utilities

### Migrating a Character Set Using the CSALTER Script

The `CSALTER` script is part of the Database Character Set Scanner utility. The `CSALTER` script is the most straightforward way to migrate a character set, but it can be used only under special circumstances: the database character set itself and all of its schema data must be a strict subset of the new character set. The new character set is a strict superset of the current character set if:
Changing the Database Character Set of an Existing Database

- Each and every character in the current character set is available in the new character set.
- Each and every character in the current character set has the same code point value in the new character set. For example, many character sets are strict supersets of US7ASCII.

With the strict superset criteria in mind, only the metadata is converted to the new character set by the CSALTER script, with the following exception: the CSALTER script performs data conversion only on CLOB columns in the data dictionary and sample schemas that have been created by Oracle. CLOB columns that users have created may need to be handled separately. Beginning with Oracle9i, some internal fields in the data dictionary and sample schemas are stored in CLOB columns. Customers may also store data in CLOB fields. When the database character set is multibyte, then CLOB data is stored in a format that is compatible with UCS-2 data. When the database character set is single-byte, then CLOB data is stored using the database character set. Because the CSALTER script converts data only in CLOB columns in the data dictionary and sample schemas that were created by Oracle, any other CLOB columns that are created must be first exported and then dropped from the schema before the CSALTER script can be run.

To change the database character set, perform the following steps:

1. Shut down the database, using either a SHUTDOWN IMMEDIATE or a SHUTDOWN NORMAL statement.
2. Do a full backup of the database, because the CSALTER script cannot be rolled back.
3. Start up the database.
4. Run the Database Character Set Scanner utility.
   CSSCAN /AS SYSDBA FULL=Y...
5. Run the CSALTER script.
   @@CSALTER.PLB NORMAL
   SHUTDOWN IMMEDIATE; -- or SHUTDOWN NORMAL;
   STARTUP;
Using the CSALTER Script in an Oracle Real Application Clusters Environment

In an Oracle Real Application Clusters environment, ensure that no other Oracle background processes are running, with the exception of the background processes associated with the instance through which a user is connected, before attempting to issue the CSALTER script. With DBA privileges, use the following SQL statement to verify that no other Oracle background processes are running:

```sql
SELECT SID, SERIAL#, PROGRAM FROM V$SESSION;
```

Set the `CLUSTER_DATABASE` initialization parameter to `FALSE` to allow the character set change to be completed. Reset it to `TRUE` after the character set has been changed.

Migrating Character Data Using the CSALTER Script and Selective Imports

Another approach to migrating character data is to perform selective exports followed by rescanning and running the CSALTER script. This approach is most common when the subset character set is single-byte and the migration is to a multibyte character set. In this scenario, user-created CLOBs must be converted because the encoding changes from the single-byte character set to a UCS-2-compatible format which Oracle uses for storage of CLOBs regardless of the multibyte encoding. The Database Character Set Scanner identifies these columns as convertible. It is up to the user to export these columns and then drop them from the schema, rescan, and, if the remaining data is clean, run the CSALTER script. When these steps have been completed, then import the CLOB columns to the database to complete migration.

Migrating to NCHAR Datatypes

Beginning with Oracle9i, data that is stored in columns of the NCHAR datatypes is stored exclusively in a Unicode encoding regardless of the database character set.
This allows users to store Unicode in a database that does not use Unicode as the database character set.

This section includes the following topics:

- Migrating Version 8 NCHAR Columns to Oracle9i and Later
- Changing the National Character Set
- Migrating CHAR Columns to NCHAR Columns

**Migrating Version 8 NCHAR Columns to Oracle9i and Later**

In the version 8 database, Oracle introduced a national character datatype (NCHAR) that allows a second, alternate character set in addition to the database character set. The NCHAR datatypes support several fixed-width Asian character sets that were introduced to provide better performance when processing Asian character data.

Beginning with Oracle9i, the SQL NCHAR datatypes are limited to Unicode character set encoding (UTF8 and AL16UTF16). Any other version 8 character sets that were available for the NCHAR datatype, including Asian character sets such as JA16SJISFIXED are no longer supported.

The steps for migrating existing NCHAR, NVARCHAR2, and NCLOB columns to NCHAR datatypes in Oracle9i and later are as follows:

1. Export all NCHAR columns from the version 8 or Oracle8i database.
2. Drop the NCHAR columns.
3. Upgrade the database to the later release.
4. Import the NCHAR columns into the upgraded database.

The migration utility can also convert version 8 and Oracle8i NCHAR columns to NCHAR columns in later releases. A SQL NCHAR upgrade script called utlnchar.sql is supplied with the migration utility. Run it at the end of the database migration to convert version 8 and Oracle8i NCHAR columns to the NCHAR columns in later releases. After the script has been executed, the data cannot be downgraded. The only way to move back to version 8 or Oracle8i is to drop all NCHAR columns, downgrade the database, and import the old NCHAR data from a previous version 8 or Oracle8i export file. Ensure that you have a backup (export file) of version 8 or Oracle8i NCHAR data, in case you need to downgrade your database in the future.
Changing the National Character Set

To change the national character set, use the ALTER DATABASE NATIONAL CHARACTER SET statement. The syntax of the statement is as follows:

```
ALTER DATABASE [db_name] NATIONAL CHARACTER SET new_NCHAR_character_set;
```

`db_name` is optional. The character set name should be specified without quotes.

You can issue the ALTER DATABASE CHARACTER SET and ALTER DATABASE NATIONAL CHARACTER SET statements together if desired.

You can use the `n_switch.sql` script, located in `$ORACLE_HOME/rdbms/admin`, to convert the national character set between UTF8 and AL16UTF16. Run it in RESTRICTED mode.

See Also: `Oracle Database SQL Reference` for the syntax of the ALTER DATABASE NATIONAL CHARACTER SET statement

Migrating CHAR Columns to NCHAR Columns

You can change a column’s datatype definition using the following methods:

- The ALTER TABLE MODIFY statement
- Online table redefinition

The ALTER TABLE MODIFY statement has the following advantages over online table redefinition:

- Easier to use
- Fewer restrictions

Online table redefinition has the following advantages over the ALTER TABLE MODIFY statement:

- Faster for columns with a large amount of data
- Can migrate several columns at one time
- Table is available for DML during most of the migration process

See Also:
- `Oracle Database Utilities` for a description of export and import procedures
- `Oracle Database Upgrade Guide` for NCHAR migration information
Migrating to NCHAR Datatypes

- Avoids table fragmentation, which saves space and allows faster access to data.
- Can be used for migration from the CLOB datatype to the NCLOB datatype

This section contains the following topics:
- Using the ALTER TABLE MODIFY Statement to Change CHAR Columns to NCHAR Columns
- Using Online Table Redefinition to Migrate a Large Table to Unicode

Using the ALTER TABLE MODIFY Statement to Change CHAR Columns to NCHAR Columns

The ALTER TABLE MODIFY statement can be used to change table column definitions from the CHAR datatypes to NCHAR datatypes. It also converts all of the data in the column from the database character set to the NCHAR character set. The syntax of the ALTER TABLE MODIFY statement is as follows:

```
ALTER TABLE table_name MODIFY (column_name datatype);
```

If indexes have been built on the migrating column, then dropping the indexes can improve the performance of the ALTER TABLE MODIFY statement because indexes are updated when each row is updated.

The maximum column lengths for NCHAR and NVARCHAR2 columns are 2000 and 4000 bytes. When the NCHAR character set is AL16UTF16, the maximum column lengths for NCHAR and NVARCHAR2 columns are 1000 and 2000 characters, which are 2000 and 4000 bytes. If this size limit is violated during migration, then consider changing the column to the NCLOB datatype instead.

---

**Note:** CLOB columns cannot be migrated to NCLOB columns using the ALTER TABLE MODIFY statement. Use online table redefinition to change a column from the CLOB datatype to the NCLOB datatype.

**See Also:** "Using Online Table Redefinition to Migrate a Large Table to Unicode" on page 11-13

Using Online Table Redefinition to Migrate a Large Table to Unicode

It takes significant time to migrate a large table with a large number of rows to Unicode datatypes. During the migration, the column data is unavailable for both reading and updating. Online table redefinition can significantly reduce migration.
time. Using online table redefinition also allows the table to be accessible to DML during most of the migration time.

Perform the following tasks to migrate a table to Unicode datatypes using online table redefinition:

1. Use the `DBMS_REDEFINITION.CAN_REDEF_TABLE` PL/SQL procedure to verify that the table can be redefined online. For example, to migrate the `scott.emp` table, enter the following command:

   ```sql
   DBMS_REDEFINITION.CAN_REDEF_TABLE('scott','emp');
   ```

2. Create an empty interim table in the same schema as the table that is to be redefined. Create it with NCHAR datatypes as the attributes. For example, enter a statement similar to the following:

   ```sql
   CREATE TABLE int_emp(
     empno NUMBER(4),
     ename NVARCHAR2(10),
     job NVARCHAR2(9),
     mgr NUMBER(4),
     hiredate DATE,
     sal NUMBER(7,2),
     deptno NUMBER(2),
     org NVARCHAR2(10));
   ```

3. Start the online table redefinition. Enter a command similar to the following:

   ```sql
   DBMS_REDEFINITION.START_REDEF_TABLE('scott',
   'emp',
   'int_emp',
   'empno empno,
   to_nchar(ename) ename,
   to_nchar(job) job,
   mgr mgr,
   hiredate hiredate,
   sal sal,
   deptno deptno,
   to_nchar(org) org');
   ```

   If you are migrating CLOB columns to NCLOB columns, then use the `TO_NCLOB` SQL conversion function instead of the `TO_NCHAR` SQL function.

4. Create triggers, indexes, grants, and constraints on the interim table. Referential constraints that apply to the interim table (the interim table is a parent or child table of the referential constraint) must be created in `DISABLED` mode. Triggers
that are defined on the interim table are not executed until the online table redefinition process has been completed.

5. You can synchronize the interim table with the original table. If many DML operations have been applied to the original table since the online redefinition began, then execute the `DBMS_REDEFINITION.SYNC_INTERIM_TABLE` procedure. This reduces the time required for the `DBMS_REDEFINITION.FINISH_REDEF_TABLE` procedure. Enter a command similar to the following:

   `DBMS_REDEFINITION.SYNC_INTERIM_TABLE('scott', 'emp', 'int_emp');`

6. Execute the `DBMS_REDEFINITION.FINISH_REDEF_TABLE` procedure. Enter a command similar to the following:

   `DBMS_REDEFINITION.FINISH_REDEF_TABLE('scott', 'emp', 'int_emp');`

   When this procedure has been completed, the following conditions are true:
   - The original table is redefined so that it has all the attributes, indexes, constraints, grants, and triggers of the interim table.
   - The referential constraints that apply to the interim table apply to the redefined original table.

7. Drop the interim table. Enter a statement similar to the following:

   `DROP TABLE int_emp;`

The results of the online table redefinition tasks are as follows:

- The original table is migrated to Unicode columns.
- The triggers, grants, indexes, and constraints defined on the interim table after the `START_REDEF_TABLE` subprogram and before the `FINISH_REDEF_TABLE` subprogram are defined for the redefined original table. Referential constraints that apply to the interim table now apply to the redefined original table and are enabled.
- The triggers, grants, indexes, and constraints defined on the original table before redefinition are transferred to the interim table and are dropped when you drop the interim table. Referential constraints that applied to the original table before redefinition were applied to the interim table and are now disabled.
- PL/SQL procedures and cursors that were defined on the original table before redefinition are invalidated. They are automatically revalidated the next time they are used. Revalidation may fail because the table definition has changed.
Tasks to Recover Database Schema After Character Set Migration

You may need to perform additional tasks to recover a migrated database schema to its original state. Consider the issues described in Table 11–2.

### Table 11–2 Issues During Recovery of a Migrated Database Schema

<table>
<thead>
<tr>
<th>Issue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexes</td>
<td>When table columns are changed from CHAR datatypes to NCHAR datatypes by the ALTER TABLE MODIFY statement, indexes that are built on the columns are changed automatically by the database. This slows down performance for the ALTER TABLE MODIFY statement. If you drop indexes before issuing the ALTER TABLE MODIFY statement, then re-create them after migration.</td>
</tr>
<tr>
<td>Constraints</td>
<td>If you disable constraints before migration, then re-enable them after migration.</td>
</tr>
<tr>
<td>Triggers</td>
<td>If you disable triggers before migration, then re-enable them after migration.</td>
</tr>
<tr>
<td>Replication</td>
<td>If the columns that are migrated to Unicode datatypes are replicated across several sites, then the changes should be executed at the master definition site. Then they are propagated to the other sites.</td>
</tr>
<tr>
<td>Binary order</td>
<td>The migration from CHAR datatypes to NCHAR datatypes involves character set conversion if the database and NCHAR data have different character sets. The binary order of the same data in different encodings can be different. This affects applications that rely on binary order.</td>
</tr>
</tbody>
</table>
Character Set Scanner Utilities

The character set scanner utilities are tools for detecting and verifying valid and invalid data. The Language and Character Set File Scanner supports text files, while the Database Character Set Scanner scans data inside the database.

This chapter introduces the Language and Character Set File Scanner and the Database Character Set Scanner. The topics in this chapter include:

- The Language and Character Set File Scanner
- The Database Character Set Scanner
- Scan Modes in the Database Character Set Scanner
- Installing and Starting the Database Character Set Scanner
- Database Character Set Scanner Parameters
- Database Character Set Scanner Sessions: Examples
- Database Character Set Scanner Reports
- How to Handle Convertible or Lossy Data in the Data Dictionary
- Storage and Performance Considerations in the Database Character Set Scanner
- Database Character Set Scanner CSALTER Script
- Database Character Set Scanner Views
- Database Character Set Scanner Error Messages
The Language and Character Set File Scanner

The Language and Character Set File Scanner (LCSSCAN) is a high-performance, statistically based utility for determining the language and character set for unknown file text. It can automatically identify a wide variety of language and character set pairs. With each text, the language and character set detection engine sets up a series of probabilities, each probability corresponding to a language and character set pair. The most statistically probable pair identifies the dominant language and character set.

The purity of the text affects the accuracy of the language and character set detection. Only plain text files are accepted, so markup tags must be stripped before using the Language and Character Set File Scanner. The ideal case is literary text of one single language with no spelling or grammatical errors. These types of text may require 100 characters of data or more and can return results with a very high factor of confidence. On the other hand, some technical documents can require longer segments before they are recognized. Documents that contain a mix of languages or character sets or text like addresses, phone numbers, or programming language code may yield poor results. For example, if a document has both French and German embedded, then the accuracy of guessing either language successfully is statistically reduced. If a document has a lot of numeric data like addresses and telephone numbers, then the accuracy of guessing is compromised.

This section includes the following topics:

- Syntax of the LCSSCAN Command
- Examples: Using the LCSSCAN Command
- Getting Command-Line Help for the Language and Character Set File Scanner
- Supported Languages and Character Sets
- LCSCCAN Error Messages

Syntax of the LCSSCAN Command

Start the Language and Character Set File Scanner with the LCSSCAN command. Its syntax is as follows:

```
LCSSCAN [RESULTS=number] [BEGIN=number] [END=number] FILE=file_name
```

The parameters are described in the rest of this section.
RESULTS
The RESULTS parameter is optional.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>1</td>
</tr>
<tr>
<td>Minimum value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum value</td>
<td>3</td>
</tr>
<tr>
<td>Purpose</td>
<td>The number of language and character set pairs that are returned. They are listed in order of probability. The comparative weight of the first choice cannot be quantified. The recommended value for this parameter is the default value of 1.</td>
</tr>
</tbody>
</table>

BEGIN
The BEGIN parameter is optional.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>1</td>
</tr>
<tr>
<td>Minimum value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum value</td>
<td>Number of bytes in file</td>
</tr>
<tr>
<td>Purpose</td>
<td>The byte of the input file where LCSSCAN begins the scanning process. The default value is the first byte of the input file.</td>
</tr>
</tbody>
</table>

END
The END parameter is optional.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>End of file</td>
</tr>
<tr>
<td>Minimum value</td>
<td>3</td>
</tr>
<tr>
<td>Maximum value</td>
<td>Number of bytes in file</td>
</tr>
<tr>
<td>Purpose</td>
<td>The last byte of the input file that LCSSCAN scans. The default value is the last byte of the input file.</td>
</tr>
</tbody>
</table>

FILE
The FILE parameter is required.
Examples: Using the LCSSCAN Command

Example 12–1 Specifying Only the File Name in the LCSSCAN Command

LCSSCAN FILE=example.txt

In this example, the entire example.txt file is scanned because the BEGIN and END parameters have not been specified. One language and character set pair will be returned because the RESULTS parameter has not been specified.

Example 12–2 Specifying the RESULTS and BEGIN Parameters for LCSSCAN

LCSSCAN RESULTS=2 BEGIN=50 FILE=example.txt

The scanning process starts at the 50th byte of the file and continues to the end of the file. Two language and character set pairs will be returned.

Example 12–3 Specifying the RESULTS and END Parameters for LCSSCAN

LCSSCAN RESULTS=3 END=100 FILE=example.txt

The scanning process starts at the beginning of the file and ends at the 100th byte of the file. Three language and character set pairs will be returned.

Example 12–4 Specifying the BEGIN and END Parameters for LCSSCAN

LCSSCAN BEGIN=50 END=100 FILE=example.txt

The scanning process starts at the 50th byte and ends at the 100th byte of the file. One language and character set pair will be returned because the RESULTS parameter has not been specified.

Getting Command-Line Help for the Language and Character Set File Scanner

To obtain a summary of the Language and Character Set File Scanner parameters, enter the following command:

LCSSCAN HELP=y
The Language and Character Set File Scanner

Character Set Scanner Utilities

12-5

The resulting output shows a summary of the Language and Character Set Scanner parameters.

Supported Languages and Character Sets

The Language and Character Set File Scanner supports several character sets for each language.

When the binary values for a language match two or more encodings that have a subset/superset relationship, the subset character set is returned. For example, if the language is German and all characters are 7-bit, then US7ASCII is returned instead of WE8MSWIN1252, WE8ISO8859P15, or WE8ISO8859P1.

When the character set is determined to be UTF-8, the Oracle character set UTF8 is returned by default unless 4-byte characters (supplementary characters) are detected within the text. If 4-byte characters are detected, then the character set is reported as AL32UTF8.

See Also: "Language and Character Set Detection Support" on page A-23 for a list of supported languages and character sets

LCSCCAN Error Messages

LCD-00001 An unknown error occurred.
Cause: An error occurred accessing an internal structure.
Action: Report this error to Oracle Support.

LCD-00002 NLS data could not be loaded.
Cause: An error occurred accessing $ORACLE_HOME/nls/data.
Action: Check to make sure $ORACLE_HOME/nls/data exists and is accessible. If not found check $ORA_NLS10 directory.

LCD-00003 An error occurred while reading the profile file.
Cause: An error occurred accessing $ORACLE_HOME/nls/data.
Action: Check to make sure $ORACLE_HOME/nls/data exists and is accessible. If not found check $ORA_NLS10 directory.

LCD-00004 The beginning or ending offset has been set incorrectly.
Cause: The beginning and ending offsets must be an integer greater than 0.
Action: Change the offset to a positive number.
The Database Character Set Scanner

LCD-00005 The ending offset has been set incorrectly.
   **Cause:** The ending offset must be greater than the beginning offset.
   **Action:** Change the ending offset to be greater than the beginning offset.

LCD-00006 An error occurred when opening the input file.
   **Cause:** The file was not found or could not be opened.
   **Action:** Check the name of the file specified. Make sure the full file name is specified and that the file is not in use.

LCD-00007 The beginning offset has been set incorrectly.
   **Cause:** The beginning offset must be less than the number of bytes in the file.
   **Action:** Check the size of the file and specify a smaller beginning offset.

LCD-00008 No result was returned.
   **Cause:** Not enough text was inputted to produce a result.
   **Action:** A larger sample of text needs to be inputted to produce a reliable result.

The Database Character Set Scanner

The Database Character Set Scanner assesses the feasibility of migrating an Oracle database to a new database character set. The Database Character Set Scanner checks all character data in the database and tests for the effects and problems of changing the character set encoding. A summary report is generated at the end of the scan that shows the scope of work required to convert the database to a new character set.

Based on the information in the summary report, you can decide on the most appropriate method to migrate the database's character set. The methods are:

- Export and Import utilities
- CSALTER script
- CSALTER script with Export and Import utilities on selected tables
Conversion Tests on Character Data

The Database Character Set Scanner reads the character data and tests for the following conditions on each data cell:

- Do character code points of the data cells change when converted to the new character set?
- Can the data cells be successfully converted to the new character set?
- Will the post-conversion data fit into the current column size?

The Database Character Set Scanner reads and tests for data in CHAR, VARCHAR2, LONG, CLOB, NCHAR, NVARCHAR2, NCLOB and VARRAY columns. The Database Character Set Scanner does not perform post-conversion column size testing for LONG, CLOB, and NCLOB columns.

Scan Modes in the Database Character Set Scanner

The Database Character Set Scanner provides three modes of database scan:

- Full Database Scan
- User Scan
- Table Scan

Full Database Scan

The Database Character Set Scanner reads and verifies the character data of all tables belonging to all users in the database including the data dictionary (such as SYS and SYSTEM users), and it reports on the effects of the simulated migration to
Installing and Starting the Database Character Set Scanner

the new database character set. It scans all schema objects including stored packages, procedures and functions, and object definitions stored as part of the data dictionary.

To understand the feasibility of migrating your database to a new database character set, you need to perform a full database scan.

User Scan

The Database Character Set Scanner reads and verifies character data of all tables belonging to the specified user and reports on the effects on the tables of changing the character set.

Table Scan

The Database Character Set Scanner reads and verifies the character data of the specified tables, and reports the effects on the tables of changing the character set.

Installing and Starting the Database Character Set Scanner

This section describes how to install and start the Database Character Set Scanner. It includes the following topics:

- Access Privileges for the Database Character Set Scanner
- Installing the Database Character Set Scanner
- Starting the Database Character Set Scanner
- Creating the Database Character Set Scanner Parameter File
- Getting Online Help for the Database Character Set Scanner

Access Privileges for the Database Character Set Scanner

To use the Database Character Set Scanner, you must have DBA privileges on the Oracle database.

Installing the Database Character Set Scanner

Before using the Database Character Set Scanner, you must run the csminst.sql script on the database that you plan to scan. The csminst.sql script needs to be run only once. The script performs the following tasks to prepare the database for scanning:
Installing and Starting the Database Character Set Scanner

- Creates a user named CSMIG
- Assigns the necessary privileges to CSMIG
- Assigns the default tablespace to CSMIG
- Connects as CSMIG
- Creates the Character Set Scanner system tables under CSMIG

The SYSTEM tablespace is assigned to CSMIG by default. Ensure that there is sufficient storage space available in the SYSTEM tablespace before scanning the database. The amount of space required depends on the type of scan and the nature of the data in the database.

**See Also:** "Storage and Performance Considerations in the Database Character Set Scanner" on page 12-38

You can modify the default tablespace for CSMIG by editing the csminst.sql script. Modify the following statement in csminst.sql to assign the preferred tablespace to CSMIG as follows:

```
ALTER USER csmig DEFAULT TABLESPACE tablespace_name;
```

On UNIX platforms, run csminst.sql using these commands and SQL statement:

```
% cd $ORACLE_HOME/rdbms/admin
% sqlplus "system/manager as sysdba"
SQL> START csminst.sql
```

**Starting the Database Character Set Scanner**

You can start the Database Character Set Scanner from the command line by one of these methods:

- Using the Database Character Set Scanner parameter file
  ```
  CSSCAN username/password PARFILE=file_name
  ```

- Using the command line to specify parameter values. For example:
  ```
  CSSCAN username/password FULL=y TOCHAR=a132utf8 ARRAY=10240 PROCESS=3
  ```

- Using an interactive session
  ```
  CSSCAN username/password
  ```
In an interactive session, the Database Character Set Scanner prompts you for the values of the following parameters:

- FULL/TABLE/USER
- TOCHAR
- ARRAY
- PROCESS

If you want to specify other parameters, then use the Database Character Set Parameter file or specify the parameter values in the CSSCAN command.

**Creating the Database Character Set Scanner Parameter File**

The Database Character Set Scanner parameter file enables you to specify Database Character Set Scanner parameters in a file where they can be easily modified or reused. Create a parameter file using a text editor.

Use one of the following formats to specify parameters in the Database Character Set Scanner parameter file:

```
parameter_name=value
parameter_name=(value1, value2, ...)
```

You can add comments to the parameter file by preceding them with the pound sign (#). All characters to the right of the pound sign are ignored.

The following is an example of a parameter file:

```
USERID=sysadmin
USER=HR # scan HR tables
TOCHAR=al32utf8
ARRAY=4096000
PROCESS=2 # use two concurrent scan processes
FEEDBACK=1000
```

**See Also:** "Database Character Set Scanner Parameters" on page 12-11

**Getting Online Help for the Database Character Set Scanner**

The Database Character Set Scanner provides online help. Enter the following command:

```
CSSCAN HELP=Y
```
The resulting output shows a summary of the Database Character Set Scanner parameters.

## Database Character Set Scanner Parameters

The following table shows a summary of parameters for the Database Character Set Scanner. The rest of this section contains detailed descriptions of the parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Prompt</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERID</td>
<td>-</td>
<td>yes</td>
<td>Username/password</td>
</tr>
<tr>
<td>FULL</td>
<td>N</td>
<td>yes</td>
<td>Scan entire database</td>
</tr>
<tr>
<td>USER</td>
<td>-</td>
<td>yes</td>
<td>Owner of the tables to be scanned</td>
</tr>
<tr>
<td>TABLE</td>
<td>-</td>
<td>yes</td>
<td>List of tables to scan</td>
</tr>
<tr>
<td>EXCLUDE</td>
<td>-</td>
<td>no</td>
<td>List of tables to exclude</td>
</tr>
<tr>
<td>TOCHAR</td>
<td>-</td>
<td>yes</td>
<td>New database character set name</td>
</tr>
<tr>
<td>FROMCHAR</td>
<td>-</td>
<td>no</td>
<td>Current database character set name</td>
</tr>
<tr>
<td>TONCHAR</td>
<td>-</td>
<td>no</td>
<td>New national character set name</td>
</tr>
<tr>
<td>FROMNCHAR</td>
<td>-</td>
<td>no</td>
<td>Current national character set name</td>
</tr>
<tr>
<td>ARRAY</td>
<td>102400</td>
<td>yes</td>
<td>Size of array fetch buffer</td>
</tr>
<tr>
<td>PROCESS</td>
<td>1</td>
<td>yes</td>
<td>Number of concurrent scan processes</td>
</tr>
<tr>
<td>MAXBLOCKS</td>
<td>-</td>
<td>no</td>
<td>The maximum number of blocks that can be in a table without the table being split</td>
</tr>
<tr>
<td>CAPTURE</td>
<td>N</td>
<td>no</td>
<td>Capture convertible data</td>
</tr>
<tr>
<td>SUPPRESS</td>
<td>-</td>
<td>no</td>
<td>Maximum number of exceptions logged for each table</td>
</tr>
<tr>
<td>FEEDBACK</td>
<td>-</td>
<td>no</td>
<td>Report progress every $n$ rows</td>
</tr>
<tr>
<td>BOUNDARIES</td>
<td>-</td>
<td>no</td>
<td>List of column size boundaries for summary report</td>
</tr>
<tr>
<td>LASTRPT</td>
<td>N</td>
<td>no</td>
<td>Generate report of the previous database scan</td>
</tr>
<tr>
<td>LOG</td>
<td>scan</td>
<td>no</td>
<td>Base file name for report files</td>
</tr>
<tr>
<td>PARFILE</td>
<td>-</td>
<td>no</td>
<td>Parameter file name</td>
</tr>
<tr>
<td>PRESERVE</td>
<td>N</td>
<td>no</td>
<td>Preserve existing scan results</td>
</tr>
<tr>
<td>LCSD</td>
<td>N</td>
<td>no</td>
<td>Enable language and character set detection</td>
</tr>
</tbody>
</table>
### Array Size Calculation

The following formula estimates the number of rows fetched at one time for a given table:

\[
\text{rows fetched} = \frac{\text{ARRAY value}}{[(\text{sum of all the CHAR and VARCHAR2 column sizes}) + (\text{number of CLOB columns \times 4000}) + (\text{number of VARRAY columns \times 4000})]}
\]

For example, suppose table A contains two CHAR columns (5 bytes and 10 bytes), two VARCHAR2 columns (100 bytes and 200 bytes), and one CLOB column. If ARRAY=102400 (the default), then the number of rows fetched is calculated as follows:

\[
102400/[5 + 10 + 100 + 200 + (1\times4000) + (0\times4000)] = 23.7
\]

The Database Character Set Scanner can fetch 23 rows of data at one time from table A.

If the sum in the denominator exceeds the value of the ARRAY parameter, then the Database Character Set Scanner fetches only one row at a time. Tables with LONG columns are fetched only one row at a time.

This parameter affects the duration of a database scan. In general, the larger the size of the array buffer, the shorter the duration time. Each scan process allocates the specified size of array buffer.

---

### Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Prompt</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCSDDATA</td>
<td>LOSSY</td>
<td>no</td>
<td>Define the scope of the language and character set detection</td>
</tr>
<tr>
<td>HELP</td>
<td>N</td>
<td>no</td>
<td>Show help screen</td>
</tr>
</tbody>
</table>

**ARRAY**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>102400</td>
</tr>
<tr>
<td>Minimum value</td>
<td>4096</td>
</tr>
<tr>
<td>Maximum value</td>
<td>Unlimited</td>
</tr>
<tr>
<td>Purpose</td>
<td>Specifies the size in bytes of the array buffer used to fetch data. The size of the array buffer determines the number of rows fetched by the Database Character Set Scanner at any one time.</td>
</tr>
</tbody>
</table>
BOUNDARIES

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>None</td>
</tr>
<tr>
<td>Purpose</td>
<td>Specifies the list of column boundary sizes that are used for an application data conversion summary report. This parameter is used to locate the distribution of the application data for the CHAR, VARCHAR2, NCHAR, and NVARCHAR2 datatypes.</td>
</tr>
</tbody>
</table>

For example, if you specify a BOUNDARIES value of (10, 100, 1000), then the application data conversion summary report produces a breakdown of the CHAR data into the following groups by their column length, CHAR(1..10), CHAR(11..100) and CHAR(101..1000). The behavior is the same for the VARCHAR2, NCHAR, and NVARCHAR2 datatypes.

CAPTURE

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>N</td>
</tr>
<tr>
<td>Range of values</td>
<td>Y or N</td>
</tr>
<tr>
<td>Purpose</td>
<td>Indicates whether to capture the information on the individual convertible rows as well as the default or storing the exception rows. The convertible rows information is written to the CSM$ERRORS table if the CAPTURE parameter is set to Y. This information can be used to deduce which records need to be converted to the target character set by using export and import processes on selected tables. When CAPTURE=Y, performance may be affected and more storage space may be required.</td>
</tr>
</tbody>
</table>

EXCLUDE

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>None</td>
</tr>
<tr>
<td>Purpose</td>
<td>Specifies the names of the tables to be excluded from the scan</td>
</tr>
</tbody>
</table>

Character Set Scanner Utilities  12-13
When this parameter is specified, the Database Character Set Scanner excludes the specified tables from the scan. You can specify the following when you specify the name of the table:

- `schemaname` specifies the name of the user’s schema from which to exclude the table
- `tablename` specifies the name of the table or tables to be excluded

For example, the following command scans all of the tables that belong to the hr sample schema except for the employees and departments tables:

```
CSSCAN system/manager USER=HR EXCLUDE=(HR.EMPLOYEES, HR.DEPARTMENTS) ...
```

### FEEDBACK

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>None</td>
</tr>
<tr>
<td>Minimum value</td>
<td>100</td>
</tr>
<tr>
<td>Maximum value</td>
<td>100000</td>
</tr>
<tr>
<td>Purpose</td>
<td>Specifies that the Database Character Set Scanner should display a progress meter in the form of a dot for every N number of rows scanned</td>
</tr>
</tbody>
</table>

For example, if you specify \texttt{FEEDBACK=1000}, then the Database Character Set Scanner displays a dot for every 1000 rows scanned. The \texttt{FEEDBACK} value applies to all tables being scanned. It cannot be set for individual tables.

### FROMCHAR

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>None</td>
</tr>
<tr>
<td>Purpose</td>
<td>Specifies the current character set name for \texttt{CHAR}, \texttt{VARCHAR2}, \texttt{LONG}, and \texttt{CLOB} datatypes in the database. By default, the Database Character Set Scanner assumes the character set for these datatypes to be the database character set.</td>
</tr>
</tbody>
</table>

Use this parameter to override the default database character set definition for \texttt{CHAR}, \texttt{VARCHAR2}, \texttt{LONG}, and \texttt{CLOB} data in the database.
### FROMNCHAR

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>None</td>
</tr>
<tr>
<td>Purpose</td>
<td>Specifies the current national database character set name for NCHAR, NVARCHAR2, and NCLOB datatypes in the database. By default, the Database Character Set Scanner assumes the character set for these datatypes to be the database national character set.</td>
</tr>
</tbody>
</table>

Use this parameter to override the default database character set definition for NCHAR, NVARCHAR2, and NCLOB data in the database.

### FULL

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>N</td>
</tr>
<tr>
<td>Range of values</td>
<td>Y or N</td>
</tr>
<tr>
<td>Purpose</td>
<td>Indicates whether to perform the full database scan (that is, to scan the entire database including the data dictionary). Specify FULL=Y to scan in full database mode.</td>
</tr>
</tbody>
</table>

**See Also:** "Scan Modes in the Database Character Set Scanner" on page 12-7 for more information about full database scans

### HELP

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>N</td>
</tr>
<tr>
<td>Range of values</td>
<td>Y or N</td>
</tr>
<tr>
<td>Purpose</td>
<td>Displays a help message with the descriptions of the Database Character Set Scanner parameters</td>
</tr>
</tbody>
</table>

**See Also:** "Getting Online Help for the Database Character Set Scanner" on page 12-10
### LASTRPT

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>N</td>
</tr>
<tr>
<td>Range of values</td>
<td>Y or N</td>
</tr>
<tr>
<td>Purpose</td>
<td>Indicates whether to regenerate the Database Character Set Scanner reports based on statistics gathered from the previous database scan</td>
</tr>
</tbody>
</table>

If LASTRPT=Y is specified, then the Database Character Set Scanner does not scan the database, but creates the report files using the information left by the previous database scan session instead.

If LASTRPT=Y is specified, then only the USERID, BOUNDARIES, and LOG parameters take effect.

### LCSD

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>N</td>
</tr>
<tr>
<td>Range of values</td>
<td>Y or N</td>
</tr>
<tr>
<td>Purpose</td>
<td>Indicates whether to apply language and character set detection during scanning</td>
</tr>
</tbody>
</table>

If LCSD=Y is specified, then the Database Character Set Scanner (CSSCAN) performs language and character set detection on the data cells categorized by the LCSDATA parameter. The accuracy of the detection depends greatly on the size and the quality of the text being analyzed. The ideal case is literary text of one single language with no spelling or grammatical errors. Data cells that contain a mixture of languages or character sets or text such as addresses and names can yield poor results. When CSSCAN cannot determine the most likely language and character set, it may return up to three most likely languages and character sets for each cell. In some cases it may return none. CSSCAN ignores any data cells with less than 10 bytes of data and returns UNKNOWN for their language and character set.

The language and character set detection is a statistically-based technology, so its accuracy varies depending on the quality of the input data. The goal is to provide CSSCAN users with additional information about unknown data inside the
It is important for CSSCAN users to review the detection results and the data itself before migrating the data to another character set.

Note that language and character set detection can affect the performance of the Database Character Set Scanner, depending on the amount of data that is being analyzed.

**See Also:** "The Language and Character Set File Scanner" on page 12-2

### LCSDDATA

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>LOSSY</td>
</tr>
<tr>
<td>Range of values</td>
<td>LOSSY, TRUNCATION, CONVERTIBLE, CHANGELESS, ALL</td>
</tr>
<tr>
<td>Purpose</td>
<td>Specifies the scope of the language and character set detection. The default is to apply the detection to only the LOSSY data cells.</td>
</tr>
</tbody>
</table>

This parameter takes effect only when LCSD=Y is specified. For example, if LCSD=Y and LCSDATA=LOSSY, CONVERTIBLE, then the Database Character Set Scanner tries to detect the character sets and languages of the data cells that are either LOSSY or CONVERTIBLE. Data that is classified as CHANGELESS and TRUNCATION will not be processed. Setting LCSDATA=ALL results in language and character set detection for all data cells scanned in the current session.

After language and character set detection has been applied to CONVERTIBLE and TRUNCATION data cells, some data cells may change from their original classification to LOSSY. This occurs when the character set detection process determines that the character set of these data cells is not the character set specified in the FROMCHAR parameter.

### LOG

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>scan</td>
</tr>
</tbody>
</table>
### Database Character Set Scanner Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| Purpose  | Specifies a base file name for the following Database Character Set Scanner report files:  
|          | - Database Scan Summary Report file, whose extension is `.txt`  
|          | - Individual Exception Report file, whose extension is `.err`  
|          | - Screen log file, whose extension is `.out` |

By default, the Database Character Set Scanner generates the three text files, `scan.txt`, `scan.err`, and `scan.out` in the current directory.

### MAXBLOCKS

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>None</td>
</tr>
<tr>
<td>Minimum value</td>
<td>1000</td>
</tr>
<tr>
<td>Maximum value</td>
<td>Unlimited</td>
</tr>
<tr>
<td>Purpose</td>
<td>Specifies the maximum block size for each table, so that large tables can be split into smaller chunks for the Database Character Set Scanner to process</td>
</tr>
</tbody>
</table>

For example, if the `MAXBLOCKS` parameter is set to 1000, then any tables that are greater than 1000 blocks in size are divided into n chunks, where  
\[ n = \text{CEIL}(\text{table block size}/1000). \]

Dividing large tables into smaller pieces is beneficial only when the number of processes set with `PROCESS` is greater than 1. If the `MAXBLOCKS` parameter is not set, then the Database Character Set Scanner attempts to split up large tables based on its own optimization rules.

### PARFILE

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>None</td>
</tr>
<tr>
<td>Purpose</td>
<td>Specifies the name for a file that contains a list of Database Character Set Scanner parameters</td>
</tr>
</tbody>
</table>
See Also: "Starting the Database Character Set Scanner" on page 12-9

PRESERVE

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>N</td>
</tr>
<tr>
<td>Range of values</td>
<td>Y or N</td>
</tr>
<tr>
<td>Purpose</td>
<td>Indicates whether to preserve the statistics gathered from the previous scan session</td>
</tr>
</tbody>
</table>

If PRESERVE=Y is specified, then the Database Character Set Scanner preserves all of the statistics from the previous scan. It adds (if PRESERVE=Y) or overwrites (if PRESERVE=N) the new statistics for the tables being scanned in the current scan request.

PROCESS

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>1</td>
</tr>
<tr>
<td>Minimum value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum value</td>
<td>32</td>
</tr>
<tr>
<td>Purpose</td>
<td>Specifies the number of concurrent scan processes to utilize for the database scan</td>
</tr>
</tbody>
</table>

SUPPRESS

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>Unset (results in unlimited number of rows)</td>
</tr>
<tr>
<td>Minimum value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum value</td>
<td>Unlimited</td>
</tr>
<tr>
<td>Purpose</td>
<td>Specifies the maximum number of data exceptions being logged for each table</td>
</tr>
</tbody>
</table>
The Database Character Set Scanner inserts information into the `CSM$ERRORS` table when an exception is found in a data cell. The table grows depending on the number of exceptions reported.

This parameter is used to suppress the logging of individual exception information after a specified number of exceptions are inserted for each table. For example, if `SUPPRESS` is set to 100, then the Database Character Set Scanner records a maximum of 100 exception records for each table.

**See Also:** "Storage Considerations for the Database Character Set Scanner" on page 12-38

### TABLE

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>None</td>
</tr>
<tr>
<td>Purpose</td>
<td>Specifies the names of the tables to scan</td>
</tr>
</tbody>
</table>

You can specify the following when you specify the name of the table:

- `schemaname` specifies the name of the user’s schema from which to scan the table
- `tablename` specifies the name of the table or tables to be scanned

For example, the following command scans the `employees` and `departments` tables in the `HR` sample schema:

```sql
CSSCAN system/manager TABLE=(HR.EMPLOYEES, HR.DEPARTMENTS)
```

### TOCHAR

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>None</td>
</tr>
<tr>
<td>Purpose</td>
<td>Specifies a target database character set name for the <code>CHAR</code>, <code>VARCHAR2</code>, <code>LONG</code>, and <code>CLOB</code> data</td>
</tr>
</tbody>
</table>
**TONCHAR**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>None</td>
</tr>
<tr>
<td>Purpose</td>
<td>Specifies a target database character set name for the NCHAR, NVARCHAR2, and NCLOB data</td>
</tr>
</tbody>
</table>

If you do not specify a value for TONCHAR, then the Database Character Set Scanner does not scan NCHAR, NVARCHAR2, and NCLOB data.

**USER**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>None</td>
</tr>
<tr>
<td>Purpose</td>
<td>Specifies the owner of the tables to be scanned</td>
</tr>
</tbody>
</table>

If the USER parameter is specified, then the Database Character Set Scanner scans all tables belonging to the specified owner. For example, the following statement scans all tables belonging to HR:

```bash
CSSCAN system/manager USER=hr ...
```

**USERID**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value</td>
<td>None</td>
</tr>
<tr>
<td>Purpose</td>
<td>Specifies the username and password (and optional connect string) of the user who scans the database. If you omit the password, then the Database Character Set Scanner prompts you for it</td>
</tr>
</tbody>
</table>

The following formats are all valid:

- `username/password`
- `username/password@connect_string`
- `username`
- `username@connect_string`
Database Character Set Scanner Sessions: Examples

The following examples show you how to use the command-line and parameter-file methods for the Full Database, User, and Table scan modes.

Full Database Scan: Examples

The following examples show how to scan the full database to see the effects of migrating it to AL32UTF8. This example assumes that the current database character set is WE8ISO8859P1.

Example: Parameter-File Method

```bash
% csscan system/manager parfile=param.txt
```

The `param.txt` file contains the following information:

- `full=y`
- `tochar=al32utf8`
- `array=4096000`
- `process=4`

Example: Command-Line Method

```bash
% csscan system/manager full=y tochar=al32utf8 array=4096000 process=4
```

Database Character Set Scanner Messages

The `scan.out` file shows which tables were scanned. The default file name for the report can be changed by using the `LOG` parameter.

See Also: "LOG" on page 12-17
. process 2 scanning SYS.TAB$[AAAAACABAAAAA0AAA]
. process 2 scanning SYS.CLU$[AAAAACABAAAAA0AAA]
. process 2 scanning SYS.ICOL$[AAAAACABAAAAA0AAA]
. process 2 scanning SYS.COL$[AAAAACABAAAAA0AAA]
. process 1 scanning SYS.IND$[AAAAACABAAAAA0AAA]
. process 1 scanning SYS.TYPE_MISC$[AAAAACABAAAAA0AAA]
. process 1 scanning SYS.LOB$[AAAAACABAAAAA0AAA]
.
.
.
. process 1 scanning IX.AQ$_ORDERS_QUEUETABLE_G
. process 2 scanning IX.AQ$_ORDERS_QUEUETABLE_I

Creating Database Scan Summary Report...

Creating Individual Exception Report...

Scanner terminated successfully.

User Scan: Examples

The following example shows how to scan the user tables to see the effects of migrating them to AL32UTF8. This example assumes the current database character set is US7ASCII, but the actual data stored is in Western European WE8MSWIN1252 encoding.

Example: Parameter-File Method

% csscan system/manager parfile=param.txt

The param.txt file contains the following information:

user=hr
fromchar=we8mswin1252
tochar=al32utf8
array=4096000
process=1

Example: Command-Line Method

% csscan system/manager user=hr fromchar=we8mswin1252 tochar=al32utf8
array=4096000 process=1
Database Character Set Scanner Sessions: Examples

Database Character Set Scanner Messages
The `scan.out` file shows which tables were scanned.

Database Character Set Scanner v2.0: Release 10.1.0.0 - Production
(c) Copyright 2003 Oracle Corporation. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.0 - Production
With the Objects option
PL/SQL Release 10.1.0.0 - Production

Enumerating tables to scan...

. process 1 scanning HR.JOBS
. process 1 scanning HR.DEPARTMENTS
. process 1 scanning HR.JOB_HISTORY
. process 1 scanning HR.EMPLOYEES

Creating Database Scan Summary Report...

Creating Individual Exception Report...

Scanner terminated successfully.

Single Table Scan: Examples

The following example shows how to scan a single table to see the effects of migrating it to WE8MSWIN1252. This example assumes the current database character set is in US7ASCII. Language and character set detection is performed on the LOSSY data cells.

Example: Parameter-File Method

```
% csscan system/manager parfile=param.txt
```

The `param.txt` file contains the following information:

```
table=hr.employees
tochar=we8mswin1252
array=4096000
process=1
suppress=100
lcsd=y
```
Example: Command-Line Method
% csscan system/manager table=hr.employees tochar=we8mswin1252 array=4096000
process=1 suppress=100 lcsd=y

Database Character Set Scanner Messages
The scan.out file shows which tables were scanned.

Database Character Set Scanner v2.0: Release 10.1.0.0 - Production
(c) Copyright 2003 Oracle Corporation. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.0 - Production
With the Objects option
PL/SQL Release 10.1.0.0 - Production

. process 1 scanning HR.EMPLOYEES

Creating Database Scan Summary Report...

Creating Individual Exception Report...

Scanner terminated successfully.

Database Character Set Scanner Reports
The Database Character Set Scanner generates two reports for each scan:
- Database Scan Summary Report
- Database Scan Individual Exception Report

The Database Scan Summary Report is found in the scan.txt file. The Database Scan Individual Exception Report is found in the scan.err file.
The default file names for the reports can be changed by using the LOG parameter.

See Also: "LOG" on page 12-17

Database Scan Summary Report
The scan.txt file contains the Database Scan Summary Report. The following output is an example of the report header. This section contains the time when each process of the scan was performed.
Database Scan Summary Report

Time Started : 2002-12-16 20:35:56
Time Completed: 2002-12-16 20:37:31

<table>
<thead>
<tr>
<th>Process ID</th>
<th>Time Started</th>
<th>Time Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2002-12-16 20:36:07</td>
<td>2002-12-16 20:37:30</td>
</tr>
<tr>
<td>2</td>
<td>2002-12-16 20:36:07</td>
<td>2002-12-16 20:37:30</td>
</tr>
</tbody>
</table>

The report consists of the following sections:

- **Database Size**
- **Database Scan Parameters**
- **Scan Summary**
- **Data Dictionary Conversion Summary**
- **Application Data Conversion Summary**
- **Application Data Conversion Summary Per Column Size Boundary**
- **Distribution of Convertible Data Per Table**
- **Distribution of Convertible Data Per Column**
- **Indexes To Be Rebuilt**
- **Truncation Due To Character Semantics**

The information available for each section depends on the type of scan and the parameters you select.

**Database Size**

This section reports on the current database size as well as identifying the amount of potential data expansion after the character set migration.

The following output is an example.

<table>
<thead>
<tr>
<th>Tablespace</th>
<th>Used</th>
<th>Free</th>
<th>Total</th>
<th>Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
<td>206.63M</td>
<td>143.38M</td>
<td>350.00M</td>
<td>588.00K</td>
</tr>
<tr>
<td>SYSAUX</td>
<td>8.25M</td>
<td>131.75M</td>
<td>140.00M</td>
<td>.00K</td>
</tr>
<tr>
<td>Total</td>
<td>214.88M</td>
<td>275.13M</td>
<td>490.00M</td>
<td>588.00K</td>
</tr>
</tbody>
</table>
The size of the largest CLOB is 57370 bytes

Database Scan Parameters
This section describes the parameters selected and the type of scan you chose. The following output is an example.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSSCAN Version</td>
<td>v2.0</td>
</tr>
<tr>
<td>Instance Name</td>
<td>rdbms06</td>
</tr>
<tr>
<td>Database Version</td>
<td>10.1.0.0.0</td>
</tr>
<tr>
<td>Scan type</td>
<td>Full database</td>
</tr>
<tr>
<td>Scan CHAR data?</td>
<td>YES</td>
</tr>
<tr>
<td>Database character set</td>
<td>WE8ISO8859P1</td>
</tr>
<tr>
<td>FROMCHAR</td>
<td>WE8ISO8859P1</td>
</tr>
<tr>
<td>TOCHAR</td>
<td>AL32UTF8</td>
</tr>
<tr>
<td>Array fetch buffer size</td>
<td>102400</td>
</tr>
<tr>
<td>Number of processes</td>
<td>2</td>
</tr>
<tr>
<td>Capture convertible data?</td>
<td>NO</td>
</tr>
<tr>
<td>Charset Language Detections</td>
<td>Yes</td>
</tr>
<tr>
<td>Charset Language Parameter</td>
<td>LOSSY</td>
</tr>
</tbody>
</table>

Scan Summary
This section indicates the feasibility of the database character set migration. There are two basic criteria that determine the feasibility of the character set migration of the database. One is the condition of the data dictionary and the other is the condition of the application data.

See Also: "Database Character Set Scanner CSALTER Script" on page 12-41

The Scan Summary section consists of two status lines: one line reports on the data dictionary, and the other line reports on the application data.

The following is sample output from the Scan Summary:

All character type data in the data dictionary are convertible to the new character set
Some character type application data are not convertible to the new character set

Table 12–1 shows the types of status that can be reported for the data dictionary and application data.
When all data remains the same in the new character set, it means that the encoding values of the original character set are identical in the target character set. For example, ASCII data is stored using the same binary values in both WE8ISO8859P1 and AL32 UTF8. In this case, the database character set can be migrated using the CSALTER script.

If all the data is convertible to the new character set, then the data can be safely migrated using the Export and Import utilities. However, the migrated data will have different encoding values in the target character set.

See Also:

- "Database Scan Individual Exception Report" on page 12-33 for more information about non-convertible data
- "Migrating a Character Set Using the CSALTER Script" on page 11-8
- "Migrating Character Data Using a Full Export and Import" on page 11-8

Data Dictionary Conversion Summary
This section contains the statistics about the conversion of the data in the data dictionary. The number of data cells with each type of status are reported by datatype. The data dictionary conversion summary is available only when a full database scan is performed.

Table 12–2 describes the possible types of status of a data cell.

### Table 12–1 Possible Status of the Data Dictionary and Application Data

<table>
<thead>
<tr>
<th>Data Dictionary Status</th>
<th>Application Data Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>All character-type data in the data dictionary remains</td>
<td>All character-type application data remains the same in the</td>
</tr>
<tr>
<td>the same in the new character set</td>
<td>new character set</td>
</tr>
<tr>
<td>All character-type data in the data dictionary is</td>
<td>All character-type application data is convertible to the</td>
</tr>
<tr>
<td>convertible to the new character set</td>
<td>new character set</td>
</tr>
<tr>
<td>Some character-type data in the data dictionary is not</td>
<td>Some character-type application data is not convertible</td>
</tr>
<tr>
<td>convertible to the new character set</td>
<td>to the new character set</td>
</tr>
</tbody>
</table>

When all data remains the same in the new character set, it means that the encoding values of the original character set are identical in the target character set. For example, ASCII data is stored using the same binary values in both WE8ISO8859P1 and AL32 UTF8. In this case, the database character set can be migrated using the CSALTER script.

If all the data is convertible to the new character set, then the data can be safely migrated using the Export and Import utilities. However, the migrated data will have different encoding values in the target character set.

See Also:

- "Database Scan Individual Exception Report" on page 12-33 for more information about non-convertible data
- "Migrating a Character Set Using the CSALTER Script" on page 11-8
- "Migrating Character Data Using a Full Export and Import" on page 11-8

### Data Dictionary Conversion Summary
This section contains the statistics about the conversion of the data in the data dictionary. The number of data cells with each type of status are reported by datatype. The data dictionary conversion summary is available only when a full database scan is performed.

Table 12–2 describes the possible types of status of a data cell.

### Table 12–2 Possible Status of Data

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changeless</td>
<td>Data remains the same in the new character set</td>
</tr>
</tbody>
</table>
The following output is an example.

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Changeless</th>
<th>Convertible</th>
<th>Truncation</th>
<th>Lossy</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2</td>
<td>1,214,557</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CHAR</td>
<td>967</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LONG</td>
<td>88,657</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CLOB</td>
<td>138</td>
<td>530</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>VARRAY</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TOTAL</td>
<td>1,304,337</td>
<td>530</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TOTAL IN PERCENTAGE</td>
<td>99.959%</td>
<td>0.041%</td>
<td>0.000%</td>
<td>0.000%</td>
</tr>
</tbody>
</table>

The data dictionary can be safely migrated using the CSALTER script.

If the number of data cells recorded in the Convertible, Truncation, and Lossy columns are zero, then no data conversion is required to migrate the data dictionary from the FROMCHAR character set to the TOCHAR character set.

If the numbers in the Truncation and Lossy columns are zero and some numbers in the Convertible columns are not zero, then all data in the data dictionary is convertible to the new character set. However, it is dangerous to convert the data in the data dictionary without understanding their impact on the database. The CSALTER script can convert some of the convertible cells in the data dictionary. The message that follows the conversion summary table indicates whether this conversion can be supported by the CSALTER script.

If the numbers in the Lossy column are not zero, then there is data in the data dictionary that is not convertible. Therefore, it is not feasible to migrate the current database to the new character because the export and import process cannot convert the data into the new character set. For example, you might have a table name with invalid characters or a PL/SQL procedure with a comment line that includes data that cannot be mapped to the new character set. These changes to schema objects must be corrected manually before migration to a new character set.

If the numbers in the Truncation column are not zero, then the export and import process would truncate the data.
See Also:

- "Database Character Set Scanner CSALTER Script" on page 12-41
- "How to Handle Convertible or Lossy Data in the Data Dictionary" on page 12-36

Application Data Conversion Summary

This section contains the statistics on conversion summary of the application data. The number of data cells with each type of status are reported by datatype. Table 12–2 describes the types of status that can be reported.

The following output is an example.

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Changeless</th>
<th>Convertible</th>
<th>Truncation</th>
<th>Lossy</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2</td>
<td>37,757</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CHAR</td>
<td>6,404</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LONG</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CLOB</td>
<td>23</td>
<td>20</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>VARRAY</td>
<td>319</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>44,507</strong></td>
<td><strong>23</strong></td>
<td><strong>0</strong></td>
<td><strong>1</strong></td>
</tr>
</tbody>
</table>
| **Total in percentage** | **99.946%** | **0.051%** | **0%**     | **0.002%**

Application Data Conversion Summary Per Column Size Boundary

This section contains the conversion summary of the CHAR and VARCHAR2 application data. The number of data cells with each type of status are reported by column size boundaries specified by the BOUNDARIES parameter. Table 12–2 describes the possible types of status.

This information is available only when the BOUNDARIES parameter is specified.

The following output is an example.

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Changeless</th>
<th>Convertible</th>
<th>Truncation</th>
<th>Lossy</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2(1..30)</td>
<td>28,702</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>VARCHAR2(31..4000)</td>
<td>9,055</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CHAR(1..30)</td>
<td>6,404</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CHAR(31..4000)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>44,161</strong></td>
<td><strong>3</strong></td>
<td><strong>0</strong></td>
<td><strong>0</strong></td>
</tr>
</tbody>
</table>

See Also:

12-30  Oracle Database Globalization Support Guide
Distribution of Convertible Data Per Table

This section shows how Convertible, Truncation, and Lossy data are distributed within the database. The statistics are reported by table. If the list contains only a few rows, then the Convertible data is localized. If the list contains many rows, then the Convertible data occurs throughout the database.

The following output is an example.

<table>
<thead>
<tr>
<th>USER.TABLE</th>
<th>Convertible</th>
<th>Truncation</th>
<th>Lossy</th>
</tr>
</thead>
<tbody>
<tr>
<td>HR.EMPLOYEES</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OE.CUSTOMERS</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM.ONLINE_MEDIA</td>
<td>13</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM.PRINT_MEDIA</td>
<td>7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SYS.EXTERNAL_TAB$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SYS.METASTYLESHEET</td>
<td>80</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Distribution of Convertible Data Per Column

This section shows how Convertible, Truncation, and Lossy data are distributed within the database. The statistics are reported by column. The following output is an example.

<table>
<thead>
<tr>
<th>USER.TABLE</th>
<th>COLUMN</th>
<th>Convertible</th>
<th>Truncation</th>
<th>Lossy</th>
</tr>
</thead>
<tbody>
<tr>
<td>HR.EMPLOYEES</td>
<td>FIRST_NAME</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OE.CUSTOMERS</td>
<td>CUST_EMAIL</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OE.CUSTOMERS</td>
<td>CUST_FIRST_NAME</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM.ONLINE_MEDIA</td>
<td>SYS_NC00042$</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM.ONLINE_MEDIA</td>
<td>SYS_NC00062$</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PM.PRINT_MEDIA</td>
<td>AD_FINALTEXT</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>PM.PRINT_MEDIA</td>
<td>AD_SOURCETEXT</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SYS.EXTERNAL_TAB$</td>
<td>PARAM_CLOB</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SYS.METASTYLESHEET</td>
<td>STYLESHEET</td>
<td>80</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Indexes To Be Rebuilt

This generates a list of all the indexes that are affected by the database character set migration. These can be rebuilt after the data has been imported. The following output is an example.

<table>
<thead>
<tr>
<th>USER.INDEX on USER.TABLE(COLUMN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HR.EMP_NAME_IX on HR.EMPLOYEES(FIRST_NAME)</td>
</tr>
<tr>
<td>HR.EMP_NAME_IX on HR.EMPLOYEES(LAST_NAME)</td>
</tr>
<tr>
<td>OE.CUST_EMAIL_IX on OE.CUSTOMERS(CUST_EMAIL)</td>
</tr>
</tbody>
</table>
**Truncation Due To Character Semantics**

This section applies only to columns that are defined using character semantics. The Truncation Due to Character Semantics section identifies the number of data cells that would be truncated if they were converted to the target character set (for example, by the SQL `CONVERT` function or another inline conversion process) before the database character set is updated with the `CSALTER` script. If the data conversion occurs after the database character set is changed, then this section can be ignored.

For example, a `VARCHAR2(5 char)` column in a WE8MSWIN1252 database can store up to 5 characters, using 5 bytes. When these characters are migrated to AL32UTF8, the same 5 characters can expand to as much as 15 bytes in length. Because the physical byte limits allocated for the column are determined by the current database character set, this column must be manually expanded to 15 bytes before the data can be converted in the target character set. Alternatively, you can apply the character set conversion to this column after the database character set has been changed. Then the same `VARCHAR2(5 char)` definition will automatically allocate 15 bytes, and no special handling is required.

The following output is an example of the Truncation Due To Character Semantics section of the report.

<table>
<thead>
<tr>
<th>USER</th>
<th>TABLE</th>
<th>COLUMN</th>
<th>Truncation</th>
</tr>
</thead>
<tbody>
<tr>
<td>HR</td>
<td>EMPLOYEES</td>
<td>FIRST_NAME</td>
<td>1</td>
</tr>
</tbody>
</table>

**Character Set Detection Result**

This section appears when the language and character set detection is turned on by the `LCSD` parameter. It displays a list of character sets detected by the Database Character Set Scanner. The character sets are ordered by occurrence. `NUMBER` refers to the number of data cells.

The following output is an example of the Character Set Detection Result section.

<table>
<thead>
<tr>
<th>CHARACTER SET</th>
<th>NUMBER</th>
<th>PERCENTAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>WE8MSWIN1252</td>
<td>38</td>
<td>97.436%</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>1</td>
<td>2.564%</td>
</tr>
</tbody>
</table>

---
Language Detection Result
This section appears when the language and character set detection is turned by the LCSD parameter. It displays a list of the languages detected by the Database Character Set Scanner. The languages are ordered by occurrence.

The following output is an example of the Language Detection Result Section.

<table>
<thead>
<tr>
<th>LANGUAGE</th>
<th>NUMBER</th>
<th>PERCENTAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENGLISH</td>
<td>36</td>
<td>92.308%</td>
</tr>
<tr>
<td>FRENCH</td>
<td>2</td>
<td>5.128%</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>1</td>
<td>2.564%</td>
</tr>
</tbody>
</table>

Database Scan Individual Exception Report
The scan.err file contains the Individual Exception Report. It consists of the following summaries:

- Database Scan Parameters
- Data Dictionary Individual Exceptions
- Application Data Individual Exceptions

Database Scan Parameters
This section describes the parameters and the type of scan chosen. The following output is an example.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSSCAN Version</td>
<td>v2.0</td>
</tr>
<tr>
<td>Instance Name</td>
<td>rdbms06</td>
</tr>
<tr>
<td>Database Version</td>
<td>10.1.0.0.0</td>
</tr>
<tr>
<td>Scan type</td>
<td>Full database</td>
</tr>
<tr>
<td>Scan CHAR data?</td>
<td>YES</td>
</tr>
<tr>
<td>Database character set</td>
<td>WE8ISO8859P1</td>
</tr>
<tr>
<td>FROMCHAR</td>
<td>WE8ISO8859P1</td>
</tr>
<tr>
<td>TOCHAR</td>
<td>AL32UTF8</td>
</tr>
<tr>
<td>Scan NCHAR data?</td>
<td>NO</td>
</tr>
<tr>
<td>Array fetch buffer size</td>
<td>102400</td>
</tr>
<tr>
<td>Number of processes</td>
<td>2</td>
</tr>
<tr>
<td>Capture convertible data?</td>
<td>NO</td>
</tr>
<tr>
<td>Charset Language Detection</td>
<td>Yes</td>
</tr>
<tr>
<td>Charset Language Parameter</td>
<td>LOSSY</td>
</tr>
</tbody>
</table>
Data Dictionary Individual Exceptions

This section reports on whether data dictionary data is convertible or has exceptions. There are two types of exceptions:

- exceed column size
- lossy conversion

The following output is an example for a data dictionary that contains convertible data.

User : SYS
Table : OBJ$
Column: NAME
Type : VARCHAR2(30)
Number of Exceptions : 0
Max Post Conversion Data Size: 30

<table>
<thead>
<tr>
<th>ROWID</th>
<th>Exception Type</th>
<th>Size Cell Data(first 30 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAAAABAAAikLAQQ</td>
<td>convertible</td>
<td>Aufträge</td>
</tr>
</tbody>
</table>

See Also:

- "Application Data Individual Exceptions" on page 12-34 for more information about exceptions
- "How to Handle Convertible or Lossy Data in the Data Dictionary" on page 12-36

Application Data Individual Exceptions

This report identifies the data with exceptions so that this data can be modified if necessary.

There are two types of exceptions:

- exceed column size
  
The column size should be extended if the maximum column width has been surpassed. Otherwise, data truncation occurs.
- lossy conversion
The data must be corrected before migrating to the new character set. Otherwise the invalid characters are converted to a replacement character. Replacement characters are usually specified as ? or â or as a character that is linguistically similar to the source character.

The following is an example of an individual exception report that illustrates some possible problems when changing the database character set from WE8ISO8859P1 to AL32UTF8.

User: USER1
Table: TEST
Column: NAME
Type: VARCHAR2(10)
Number of Exceptions: 2
Max Post Conversion Data Size: 11

<table>
<thead>
<tr>
<th>ROWID</th>
<th>Exception Type</th>
<th>Size</th>
<th>Cell Data(first 30 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAA2fAAFAABJwQAg</td>
<td>exceed column size</td>
<td>11</td>
<td>Ährenfeldt</td>
</tr>
<tr>
<td>AAAA2fAAFAABJwQAg</td>
<td>lossy conversion</td>
<td>11</td>
<td>órâclë8™</td>
</tr>
<tr>
<td>AAAA2fAAFAABJwQAg</td>
<td>exceed column size</td>
<td>11</td>
<td>órâclë8™</td>
</tr>
</tbody>
</table>

The values Ährenfeldt and órâclë8™ exceed the column size (10 bytes) because each of the characters Ä, ö, å, and ë occupies one byte in WE8ISO8859P1 but two bytes in AL32UTF8. The value órâclë8™ has lossy conversion to AL32UTF8 because the trademark sign ™ (code 153) is not a valid WE8ISO8859P1 character. It is a WE8MSWIN1252 character, which is a superset of WE8ISO8859P1.

You can view the data that has an exception by issuing a SELECT statement:

```
SELECT name FROM user1.test WHERE ROWID='AAAA2fAAFAABJwQAg';
```

You can modify the data that has the exception by issuing an UPDATE statement:

```
UPDATE user1.test SET name = 'Oracle8 TM'
WHERE ROWID='AAAA2fAAFAABJwQAg';
```

If the language and character set detection option is enabled, then CSSCAN attempts to provide the most probable languages and character sets for the data cells specified by the LCSDDATA parameter.

The following is an example of an Individual Exception Report that illustrates language and character set detection results for lossy data cells when changing the database character set from US7ASCII to AL32UTF8.
How to Handle Convertible or Lossy Data in the Data Dictionary

Unlike modifying user application data, updating and changing the contents of data dictionary tables directly is not supported. Updating the system tables without understanding the internal dependencies can lead to database corruption.

If the data dictionary is convertible, then the data cells are encoded in the FROMCHAR character set. Two common scenarios for the existence of convertible data in the data dictionary are:

- **CLOB data in the data dictionary**
  
  For single-byte character sets, CLOB data is stored in the database character set encoding. For multibyte character sets, CLOB data is stored in an internal Oracle format which is compatible with UCS-2. For example, the byte representation of the string 'ABC' stored inside a VARCHAR2 column in a US7ASCII database remains unchanged when migrated to AL32UTF8. The same string stored inside a CLOB column doubles in size and is stored completely differently. When migrating from a single-byte character set to a multibyte character set, CLOB data is never CHANGELESS.

- **Migrating a database to a character set that is a superset in the sense that it contains all of the characters of the original character set, but the binary values of some characters is not the same in the two character sets**

  This is similar to user application data whose data cells need to be manually converted to the new character set. A common cause of this is that the user’s

---

See Also:

- "Data Truncation" on page 11-2
- "Character Set Conversion Issues" on page 11-4
object definitions (such as table names, column names, package names and package bodies) were created using non-ASCII characters. These are typically characters or symbols that are part of the user's native language.

The easiest approach to migrating convertible data dictionary data is to create a new database in the target character set and then re-create all of the data dictionary and schema definitions by exporting and importing. However, this method means creating a new database.

If you want to migrate the existing database instead of building a new one, then the CSALTER script is the simplest way to migrate convertible CLOB data inside the data dictionary and to change the existing database character set to the target character set.

**See Also:** "Database Character Set Scanner CSALTER Script" on page 12-41

For data dictionary CONVERTIBLE data cells that are not CLOB data, you must find the schema objects containing the convertible data. Then you can choose to do one of the following:

- Amend the object definitions (such as removing non-ASCII characters from comments inside a package body) so that the data cells become CHANGELESS
- Drop these objects from the database schema altogether and then re-create them after the database character set has been migrated to the target character set

LOSSY dictionary data cells require further examination of the data dictionary to see whether the current FROMCHAR character set is the actual character set of the database. If it is, you have to correct these object definitions (such as removing the offending characters from comments inside a package body) so that the data cells become CHANGELESS and they can be migrated safely to the target character set.

Three SQL scripts are included with the Database Character Set Scanner to help users to locate the CONVERTIBLE and LOSSY data cells in the data dictionary:

- analyze_source.sql
- analyze_histgram.sql
- analyze_rule.sql

The scripts are stored in the $ORACLE_HOME/nls/csscan/sql directory. They perform SQL SELECT operations on the SYS.SOURCE$, SYS.HISTGRM$ and SYS.RULE$ data dictionary tables so that the offending data dictionary objects can be identified.
The following example shows output from the `analyze_source.sql` script:

```
SQL> @ORACLE_HOME/nls/csscan/sql/analyze_source.sql

Table: SYS.SOURCE$
Error: CONVERTIBLE DATA

no rows selected

Table: SYS.SOURCE$
Error: EXCEPTIONAL DATA

<table>
<thead>
<tr>
<th>OWNER</th>
<th>OBJECT_NAME</th>
<th>OBJECT_TYPE</th>
<th>EXCEPTIONAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCOTT</td>
<td>FOO</td>
<td>FUNCTION</td>
<td>1</td>
</tr>
</tbody>
</table>
```

## Storage and Performance Considerations in the Database Character Set Scanner

This section describes storage and performance issues in the Database Character Set Scanner. It contains the following topics:

- **Storage Considerations for the Database Character Set Scanner**
- **Performance Considerations for the Database Character Set Scanner**
- **Recommendations and Restrictions for the Database Character Set Scanner**

### Storage Considerations for the Database Character Set Scanner

This section describes the size and the growth of the Database Character Set Scanner's system tables, and explains the approach to maintain them. There are three system tables that can increase rapidly depending on the nature of the data stored in the database.

You may want to assign a large tablespace to the user CSMIG by amending the `csminst.sql` script. By default, the SYSTEM tablespace is assigned to the user CSMIG.

This section includes the following topics:

- **CSM$TABLES**
- **CSM$COLUMNS**
- **CSM$ERRORS**
CSM$TABLES
The Database Character Set Scanner enumerates all tables that need to be scanned into the CSM$TABLES table.

You can look up the number of tables (to get an estimate of how large CSM$TABLES can become) in the database by issuing the following SQL statement:

```sql
SELECT COUNT(*) FROM DBA_TABLES;
```

CSM$COLUMNS
The Database Character Set Scanner stores statistical information for each column scanned into the CSM$COLUMNS table.

You can look up the number of character type columns (to get an estimate of how large CSM$COLUMNS can become) in the database by issuing the following SQL statement:

```sql
SELECT COUNT(*) FROM DBA_TAB_COLUMNS
WHERE DATA_TYPE IN ('CHAR', 'VARCHAR2', 'LONG', 'CLOB', 'VARRAY');
```

CSM$ERRORS
When exceptions are detected with cell data, the Database Character Set Scanner inserts individual exception information into the CSM$ERRORS table. This information then appears in the Individual Exception Report and facilitates identifying records to be modified if necessary.

If your database contains a lot of data that is signaled as Exceptional or Convertible (when the parameter CAPTURE=Y is set), then the CSM$ERRORS table can grow very large. You can prevent the CSM$ERRORS table from growing unnecessarily large by using the SUPPRESS parameter.

The SUPPRESS parameter applies to all tables. The Database Character Set Scanner suppresses inserting individual Exceptional information after the specified number of exceptions is inserted. Limiting the number of exceptions to be recorded may not be useful if the exceptions are spread over different tables.

Performance Considerations for the Database Character Set Scanner
This section describes how to increase performance when scanning the database.
Using Multiple Scan Processes
If you plan to scan a relatively large database, for example, over 50GB, then you might want to consider using multiple scan processes. This shortens the duration of database scans by using hardware resources such as CPU and memory available on the machine. A guideline for determining the number of scan processes to use is to set the number equal to the CPU_COUNT initialization parameter.

Setting the Array Fetch Buffer Size
The Database Character Set Scanner fetches multiple rows at a time when an array fetch is allowed. You can usually improve performance by letting the Database Character Set Scanner use a bigger array fetch buffer. Each process allocates its own fetch buffer.

Suppressing Exception and Convertible Log
The Database Character Set Scanner inserts individual Exceptional and Convertible (when CAPTURE=Y) information into the CSM$ERRORS table. In general, insertion into the CSM$ERRORS table is more costly than data fetching. If your database has a lot of data that is signaled as Exceptional or Convertible, then the Database Character Set Scanner issues many insert statements, causing performance degradation. Oracle Corporation recommends setting a limit on the number of exception rows to be recorded using the SUPPRESS parameter.

Recommendations and Restrictions for the Database Character Set Scanner
All the character-based data in CHAR, VARCHAR2, LONG, CLOB, and VARRAY columns are stored in the database character set, which is specified with the CREATE DATABASE statement when the database is first created. However, in some configurations, it is possible to store data in a different character set from the database character set either intentionally or unintentionally. This happens most often when the NLS_LANG character set is the same as the database character set, because in such cases Oracle sends and receives data as is, without any conversion or validation. It can also happen if one of the two character sets is a superset of the other, in which case many of the code points appear as if they were not converted. For example, if NLS_LANG is set to WE8ISO8859P1 and the database character set is WE8MSWIN1252, then all code points except the range 128-159 are preserved through the client/server conversion.

The same binary code point value can be used to represent different characters between the different character sets. Most European character sets share liberal use of the 8-bit range to encode native characters, so it is very possible for a cell to be...
reported as convertible but for the wrong reasons. When you set the FROMCHAR parameter, the assumption is that all character data is encoded in that character set, but the Database Character Set Scanner may not be able to accurately determine its validity.

For example, this can occur when the Database Character Set Scanner is used with the FROMCHAR parameter set to WE8MSWIN1252. This single-byte character set encodes a character in every available code point so that no matter what data is being scanned, the scanner always identifies a data cell as being available in the source character set.

Scanning Database Containing Data Not in the Database Character Set
If a database contains data that is not in the database character set, but it is encoded in another character set, then the Database Character Set Scanner can perform a scan if the FROMCHAR parameter specifies the encoded character set.

Scanning Database Containing Data from two or more Character Sets
If a database contains data from more than one character set, then the Database Character Set Scanner cannot accurately test the effects of changing the database character set by a single scan. If the data can be divided into separate tables, one for each character set, then the Database Character Set Scanner can perform multiple table scans to verify the validity of the data.

For each scan, use a different value of the FROMCHAR parameter to tell the Database Character Set Scanner to treat all target columns in the table as if they were in the specified character set.

Database Character Set Scanner CSALTER Script

The CSALTER script is a DBA tool for special character set migration. Similar to the obsolete ALTER DATABASE CHARACTER SET SQL statement, CSALTER should be used when the migration is from an existing character set that is a proper subset of the new database character set. CSALTER should be used only by the system administrator. System administrators must run the Database Character Set Scanner first to confirm that the proper conditions exist for running CSALTER. Also, the database must be backed up before running CSALTER.

To run the CSALTER script, start SQL*Plus and connect to the database whose character set is to be migrated. Note that the Database Character Set Scanner must be run before the CSALTER script. Then enter the following command:

sqlplus> @CSALTER.PLB normal
The CSALTER script includes the following phases:

- Checking Phase of the CSALTER Script
- Updating Phase of the CSALTER Script

Checking Phase of the CSALTER Script

In the checking phase, the CSALTER script performs the following tasks:

1. It checks whether the user login is SYS. Only user SYS is allowed to run the script.
2. It checks whether a full database scan has been previously run within the last 7 days. If a full database scan has not been previously run, then the script stops and reports an error. It is the DBA's responsibility to ensure that no one updates the database between the times the full database scan and the CSALTER script are run.
3. It checks whether the new character set is a superset of the existing character set. If the new character set is not a superset of the existing one, then the script stops and reports that the CSALTER script cannot be completed.
4. It checks whether CLOB columns in the data dictionary that were created by Oracle are changeless or convertible. Convertible CLOB columns occur when migrating from a single-byte character set to a multibyte character set. If there are any lossy cells found in CLOB columns in the data dictionary, then the script stops. The lossy CLOB columns may need to be specially handled; contact Oracle Support Services for more information.

Any table that belongs to the following schemas is considered to be part of the data dictionary:

SYS
SYSTEM
CTXSYS
DIP
DMSYS
EXFSYS
LBACSYS
MDSYS
ORDPLUGINS
ORDSYS
SI_INFORMTN_SCHEMA
XDB

5. It checks whether all CLOB columns in the Sample Schemas created by Oracle are changeless or convertible. Convertible CLOB columns occur when migrating from a single-byte character set to a multibyte character set. The tables that belong to the following schemas are part of the Sample Schemas:

HR
OE
SH
PM

6. It checks whether the CLOB datatype is the only datatype that contains convertible data in the data dictionary and Sample Schemas. It checks that all other users’ tables have no convertible data for all datatypes including the CLOB datatype. Because the CSALTER script is meant to be run only when the current database is a proper subset of the new database, all data should be changeless with the possible exception of the CLOB data. When migrating from a single-byte character set to a multibyte character set, user-created CLOB data requires conversion and must first be exported and deleted from the schema. The database must be rescanned in order to run the CSALTER script. Cells of all other datatypes that are reported to be convertible or subject to truncation must be corrected before the Database Character Set Scanner is rerun.

See Also:

- "Subsets and Supersets" on page A-20
- "Migrating a Character Set Using the CSALTER Script" on page 11-8 for more information about the CSALTER script and CLOB data

Updating Phase of the CSALTER Script

After the CSALTER script confirms that every CLOB in the data dictionary passes the checks described in "Checking Phase of the CSALTER Script" on page 12-42, the CSALTER script performs the conversion. After all CLOB data in the data dictionary and the Sample Schemas have been updated, the script commits the change and saves the information in the CSM$TABLES view. After all CLOB data in the data dictionary have been updated, the CSALTER script updates the database metadata to the new character set. The entire migration process is then completed.
The **CSALTER** script is resumable. If the update of the database to the new character set fails at any time, then the DBA must shut down and restart the database and rerun the **CSALTER** script before doing anything else. Because the updated information is already saved in the **CSM$TABLES** view, the script will not update the **CLOB** data in the data dictionary tables twice. The process of migration is simply resumed to finish the update of the database to the new character set.

If the **CSALTER** script fails, then use the following method to resume the update:

1. From the SQL*Plus session where the **CSALTER** script was run, enter the following command immediately:
   ```sql
   SHUTDOWN ABORT
   ```
2. Start up the database without mounting it, because **CSALTER** requires an open database.
   ```sql
   START NOMOUNT
   ```
3. Run the **CSALTER** script:
   ```sql
   @@CSALTER.PLB NORMAL
   ```
4. Shut down the database with either the **IMMEDIATE** or the **NORMAL** option.
5. Start up the database.

### Database Character Set Scanner Views

The Database Character Set Scanner uses the following views:

- **CSMV$COLUMNS**
- **CSMV$CONSTRAINTS**
- **CSMV$ERRORS**
- **CSMV$INDEXES**
- **CSMV$TABLES**

**CSMV$COLUMNS**

This view contains statistical information about columns that were scanned.
### CSMV$CONSTRAINTS

This view contains statistical information about columns that were scanned.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNER_ID</td>
<td>NUMBER</td>
<td>NOT NULL</td>
<td>Userid of the constraint owner</td>
</tr>
<tr>
<td>OWNER_NAME</td>
<td>VARCHAR2 (30)</td>
<td>NOT NULL</td>
<td>User name of the constraint owner</td>
</tr>
<tr>
<td>CONSTRAINT_ID</td>
<td>NUMBER</td>
<td>NOT NULL</td>
<td>Object ID of the constraint</td>
</tr>
<tr>
<td>CONSTRAINT_NAME</td>
<td>VARCHAR2 (30)</td>
<td>NOT NULL</td>
<td>Object name of the constraint</td>
</tr>
<tr>
<td>CONSTRAINT_TYPE#</td>
<td>NUMBER</td>
<td>NOT NULL</td>
<td>Constraint type number</td>
</tr>
<tr>
<td>CONSTRAINT_TYPE</td>
<td>VARCHAR2 (11)</td>
<td>NOT NULL</td>
<td>Constraint type name</td>
</tr>
<tr>
<td>TABLE_ID</td>
<td>NUMBER</td>
<td>NOT NULL</td>
<td>Object ID of the table</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR2 (30)</td>
<td>NOT NULL</td>
<td>Object name of the table</td>
</tr>
<tr>
<td>CONSTRAINT_RID</td>
<td>NUMBER</td>
<td>NOT NULL</td>
<td>Root constraint ID</td>
</tr>
</tbody>
</table>
CSMV$ERRORS

This view contains individual exception information for cell data and object definitions.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRAINT_LEVEL</td>
<td>NUMBER</td>
<td>NOT NULL</td>
<td>Constraint level</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNER_ID</td>
<td>NUMBER</td>
<td>NOT NULL</td>
<td>Userid of the table owner</td>
</tr>
<tr>
<td>OWNER_NAME</td>
<td>VARCHAR2 (30)</td>
<td>NOT NULL</td>
<td>User name of the table owner</td>
</tr>
<tr>
<td>TABLE_ID</td>
<td>NUMBER</td>
<td>NOT NULL</td>
<td>Object ID of the table</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR2 (30)</td>
<td>-</td>
<td>Object name of the table</td>
</tr>
<tr>
<td>COLUMN_ID</td>
<td>NUMBER</td>
<td>-</td>
<td>Column ID</td>
</tr>
<tr>
<td>COLUMN_INTID</td>
<td>NUMBER</td>
<td>-</td>
<td>Internal column ID (for abstract datatypes)</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>VARCHAR2 (30)</td>
<td>-</td>
<td>Column name</td>
</tr>
<tr>
<td>DATA_ROWID</td>
<td>VARCHAR2 (1000)</td>
<td>-</td>
<td>The rowid of the data</td>
</tr>
<tr>
<td>COLUMN_TYPE</td>
<td>VARCHAR2 (9)</td>
<td>-</td>
<td>Column datatype of object type</td>
</tr>
<tr>
<td>ERROR_TYPE</td>
<td>VARCHAR2 (11)</td>
<td>-</td>
<td>Type of error encountered</td>
</tr>
</tbody>
</table>

CSMV$INDEXES

This view contains individual exception information for indexes.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX_OWNER_ID</td>
<td>NUMBER</td>
<td>NOT NULL</td>
<td>Userid of the index owner</td>
</tr>
<tr>
<td>INDEX_OWNER_NAME</td>
<td>VARCHAR2 (30)</td>
<td>NOT NULL</td>
<td>User name of the index owner</td>
</tr>
<tr>
<td>INDEX_ID</td>
<td>NUMBER</td>
<td>NOT NULL</td>
<td>Object ID of the index</td>
</tr>
<tr>
<td>INDEX_NAME</td>
<td>VARCHAR2 (30)</td>
<td>-</td>
<td>Object name of the index</td>
</tr>
<tr>
<td>INDEX_STATUS#</td>
<td>NUMBER</td>
<td>-</td>
<td>Status number of the index</td>
</tr>
<tr>
<td>INDEX_STATUS</td>
<td>VARCHAR2 (8)</td>
<td>-</td>
<td>Status of the index</td>
</tr>
<tr>
<td>TABLE_OWNER_ID</td>
<td>NUMBER</td>
<td>-</td>
<td>Userid of the table owner</td>
</tr>
<tr>
<td>TABLE_OWNER_NAME</td>
<td>VARCHAR2 (30)</td>
<td>-</td>
<td>User name of the table owner</td>
</tr>
</tbody>
</table>
**CSMV$TABLES**

This view contains information about database tables to be scanned. The Database Character Set Scanner enumerates all tables to be scanned into this view.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>NULL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_ID</td>
<td>NUMBER</td>
<td>–</td>
<td>Object ID of the table</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>VARCHAR2 (30)</td>
<td>–</td>
<td>Object name of the table</td>
</tr>
<tr>
<td>COLUMN_ID</td>
<td>NUMBER</td>
<td>–</td>
<td>Column ID</td>
</tr>
<tr>
<td>COLUMN_INTID</td>
<td>NUMBER</td>
<td>–</td>
<td>Internal column ID (for abstract datatypes)</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>VARCHAR2 (30)</td>
<td>–</td>
<td>Column name</td>
</tr>
</tbody>
</table>

**Database Character Set Scanner Error Messages**

The Database Character Set Scanner has the following error messages:

CSS-00100 failed to allocate memory size of number

*Cause:* An attempt was made to allocate memory with size 0 or bigger than the maximum size.

*Action:* This is an internal error. Contact Oracle Support Services.

CSS-00101 failed to release memory
Cause: An attempt was made to release memory with an invalid pointer.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00102 failed to release memory, null pointer given
Cause: An attempt was made to release memory with a null pointer.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00105 failed to parse BOUNDARIES parameter
Cause: BOUNDARIES parameter was specified in an invalid format.
Action: Refer to the manual for the correct syntax.

CSS-00106 failed to parse SPLIT parameter
Cause: SPLIT parameter was specified in an invalid format.
Action: Refer to the documentation for the correct syntax.

CSS-00107 Character set migration utility schema not installed
Cause: CSM$VERSION table not found in the database.
Action: Run CSMINST.SQL on the database.

CSS-00108 Character set migration utility schema not compatible
Cause: Incompatible CSM$* tables found in the database.
Action: Run CSMINST.SQL on the database.

CSS-00110 failed to parse userid
Cause: USERID parameter was specified in an invalid format.
Action: Refer to the documentation for the correct syntax.

CSS-00111 failed to get RDBMS version
Cause: Failed to retrieve the value of the Version of the database.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00112 database version not supported
Cause: The database version is older than release 8.0.5.0.0.
Action: Upgrade the database to release 8.0.5.0.0 or later. Then try again.

CSS-00113 user %s is not allowed to access data dictionary
Cause: The specified user cannot access the data dictionary.
Database Character Set Scanner Error Messages

Action: Set O7_DICTIONARY_ACCESSIBILITY parameter to TRUE, or use SYS user.

CSS-00114 failed to get database character set name
Cause: Failed to retrieve value of NLS_CHARACTERSET or NLS_NCHAR_CHARACTERSET parameter from NLS_DATABASE_PARAMETERS view.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00115 invalid character set name %s
Cause: The specified character set is not a valid Oracle character set.
Action: Retry with a valid Oracle character set name.

CSS-00116 failed to reset NLS_LANG/NLS_NCHAR parameter
Cause: Failed to force NLS_LANG character set to be the same as the database character set.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00117 failed to clear previous scan log
Cause: Failed to delete all rows from CSM$* tables.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00118 failed to save command parameters
Cause: Failed to insert rows into CSM$PARAMETERS table.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00119 failed to save scan start time
Cause: Failed to insert a row into CSM$PARAMETERS table.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00120 failed to enumerate tables to scan
Cause: Failed to enumerate tables to scan into CSM$TABLES table.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00121 failed to save scan complete time
Cause: Failed to insert a row into CSM$PARAMETERS table.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00122 failed to create scan report
Cause: Failed to create database scan report.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00123 failed to check if user %s exist
Cause: SELECT statement that checks if the specified user exists in the database failed.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00124 user %s not found
Cause: The specified user does not exist in the database.
Action: Check the user name.

CSS-00125 failed to check if table %s.%s exist
Cause: SELECT statement that checks if the specified table exists in the database failed.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00126 table %s.%s not found
Cause: The specified table does not exist in the database.
Action: Check the user name and table name.

CSS-00127 user %s does not have DBA privilege
Cause: The specified user does not have DBA privileges, which are required to scan the database.
Action: Choose a user with DBA privileges.

CSS-00128 failed to get server version string
Cause: Failed to retrieve the version string of the database.
Action: None.

CSS-00130 failed to initialize semaphore
Cause: Unknown.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00131 failed to spawn scan process %d
Cause: Unknown.
Action: This is an internal error. Contact Oracle Support Services.
CSS-00132 failed to destroy semaphore
   Cause: Unknown.
   Action: This is an internal error. Contact Oracle Support Services.

CSS-00133 failed to wait semaphore
   Cause: Unknown.
   Action: This is an internal error. Contact Oracle Support Services.

CSS-00134 failed to post semaphore
   Cause: Unknown.
   Action: This is an internal error. Contact Oracle Support Services.

CSS-00140 failed to scan table (tid=%d, oid=%d)
   Cause: Data scan on specified table failed.
   Action: This is an internal error. Contact Oracle Support Services.

CSS-00141 failed to save table scan start time
   Cause: Failed to update a row in the CSM$TABLES table.
   Action: This is an internal error. Contact Oracle Support Services.

CSS-00142 failed to get table information
   Cause: Failed to retrieve various information from userID and object ID of the table.
   Action: This is an internal error. Contact Oracle Support Services.

CSS-00143 failed to get column attributes
   Cause: Failed to retrieve column attributes of the table.
   Action: This is an internal error. Contact Oracle Support Services.

CSS-00144 failed to scan table %s.%s
   Cause: Data scan on specified table failed.
   Action: This is an internal error. Contact Oracle Support Services.

CSS-00145 failed to save scan result for columns
   Cause: Failed to insert rows into CSM$COLUMNS table.
   Action: This is an internal error. Contact Oracle Support Services.

CSS-00146 failed to save scan result for table
Cause: Failed to update a row of CSM$TABLES table.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00147 unexpected data truncation
Cause: Scanner allocates the exactly same size of memory as the column byte size for fetch buffer, resulting in unexpected data truncation.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00150 failed to enumerate table
Cause: Failed to retrieve the specified table information.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00151 failed to enumerate user tables
Cause: Failed to enumerate all tables that belong to the specified user.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00152 failed to enumerate all tables
Cause: Failed to enumerate all tables in the database.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00153 failed to enumerate character type columns
Cause: Failed to enumerate all CHAR, VARCHAR2, LONG, and CLOB columns of tables to scan.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00154 failed to create list of tables to scan
Cause: Failed to enumerate the tables into CSM$TABLES table.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00155 failed to split tables for scan
Cause: Failed to split the specified tables.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00156 failed to get total number of tables to scan
Cause: SELECT statement that retrieves the number of tables to scan failed.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00157 failed to retrieve list of tables to scan
Cause: Failed to read all table IDs into the scanner memory.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00158 failed to retrieve index defined on column
Cause: SELECT statement that retrieves index defined on the column failed.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00160 failed to open summary report file
Cause: FILE OPEN function returned error.
Action: Check if you have create/write privilege on the disk and check if the file name specified for the LOG parameter is valid.

CSS-00161 failed to report scan elapsed time
Cause: Unknown.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00162 failed to report database size information
Cause: Unknown.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00163 failed to report scan parameters
Cause: Unknown.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00164 failed to report scan summary
Cause: Unknown.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00165 failed to report conversion summary
Cause: Unknown.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00166 failed to report convertible data distribution
Cause: Unknown.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00167 failed to open exception report file
Cause: FILE OPEN function returned error.
Action: Check if you have create/write privilege on the disk and check if the file name specified for LOG parameter is valid.

CSS-00168 failed to report individual exceptions
Cause: Unknown.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00170 failed to retrieve size of tablespace %
Cause: Unknown.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00171 failed to retrieve free size of tablespace %s
Cause: Unknown.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00172 failed to retrieve total size of tablespace %s
Cause: Unknown.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00173 failed to retrieve used size of the database
Cause: Unknown.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00174 failed to retrieve free size of the database
Cause: Unknown.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00175 failed to retrieve total size of the database
Cause: Unknown.
Action: This is an internal error. Contact Oracle Support Services.

CSS-00176 failed to enumerate user tables in bitmapped tablespace
Cause: Failed to enumerate tables in bitmapped tablespace.
Action: This is an internal error. Contact Oracle Support Services.
This chapter shows how to customize locale data. It includes the following topics:

- Overview of the Oracle Locale Builder Utility
- Creating a New Language Definition with the Oracle Locale Builder
- Creating a New Territory Definition with the Oracle Locale Builder
- Displaying a Code Chart with the Oracle Locale Builder
- Creating a New Character Set Definition with the Oracle Locale Builder
- Creating a New Linguistic Sort with the Oracle Locale Builder
- Generating and Installing NLB Files
Overview of the Oracle Locale Builder Utility

The Oracle Locale Builder offers an easy and efficient way to customize locale data. It provides a graphical user interface through which you can easily view, modify, and define locale-specific data. It extracts data from the text and binary definition files and presents them in a readable format so that you can process the information without worrying about the formats used in these files.

The Oracle Locale Builder manages four types of locale definitions: language, territory, character set, and linguistic sort. It also supports user-defined characters and customized linguistic rules. You can view definitions in existing text and binary definition files and make changes to them or create your own definitions.

This section contains the following topics:

- Configuring Unicode Fonts for the Oracle Locale Builder
- The Oracle Locale Builder User Interface
- Oracle Locale Builder Windows and Dialog Boxes

Configuring Unicode Fonts for the Oracle Locale Builder

The Oracle Locale Builder uses Unicode characters in many of its functions. For example, it shows the mapping of local character code points to Unicode code points. Oracle Locale Builder depends on the local fonts that are available on the operating system where the characters are rendered. Therefore, Oracle Corporation recommends that you use a Unicode font to fully support the Oracle Locale Builder. If a character cannot be rendered with your local fonts, then it will probably be displayed as an empty box.

Font Configuration on Windows

There are many Windows TrueType and OpenType fonts that support Unicode. Oracle Corporation recommends using the Arial Unicode MS font from Microsoft, because it includes about 51,000 glyphs and supports most of the characters in Unicode 3.2.

After installing the Unicode font, add the font to the Java Runtime
font.properties file so it can be used by the Oracle Locale Builder. The
font.properties file is located in the $JAVAHOME/jre/lib directory. For example, for the Arial Unicode MS font, add the following entries to the
font.properties file:

dialog.n=Arial Unicode MS, DEFAULT_CHARSET
dialoginput.n=Arial Unicode MS, DEFAULT_CHARSET
Overview of the Oracle Locale Builder Utility

serif.\n=Arial Unicode MS, DEFAULT_CHARSET
sansserif.\n=Arial Unicode MS, DEFAULT_CHARSET

\n is the next available sequence number to assign to the Arial Unicode MS font in the font list. Java Runtime searches the font mapping list for each virtual font and use the first font available on your system.

After you edit the font.properties file, restart the Oracle Locale Builder.

See Also: Sun’s internationalization Web site for more information about the font.properties file

Font Configuration on Other Platforms
There are fewer choices of Unicode fonts for non-Windows platforms than for Windows platforms. If you cannot find a Unicode font with satisfactory character coverage, then use multiple fonts for different languages. Install each font and add the font entries into the font.properties file using the steps described for the Windows platform.

For example, to display Japanese characters on Sun Solaris using the font ricoh-hg mincho, add entries to the existing font.properties file in $JAVAHOME/lib in the dialog, dialoginput, serif, and sansserif sections. For example:

serif.plain.3=-ricoh-hg mincho 1-medium-r-normal--*-\d-\*-*m-\*-jisx0201.1976-0

Note: Depending on the operating system locale, the locale-specific font.properties file might be used. For example, if the current operating system locale is ja_JP.eucJP on Sun Solaris, then font.properties.ja may be used.

See Also: Your operating system documentation for more information about available fonts

The Oracle Locale Builder User Interface
Ensure that the ORACLE_HOME parameter is set before starting Oracle Locale Builder.

In the UNIX operating system, start the Oracle Locale Builder by changing into the $ORACLE_HOME/nls/lbuilder directory and issuing the following command:
% ./lbuilder

In a Windows operating system, start the Oracle Locale Builder from the Start menu as follows: Start > Programs > Oracle-OraHome10 > Configuration and Migration Tools > Locale Builder. You can also start it from the DOS prompt by entering the %ORACLE_HOME%\nls\lbuilder directory and executing the lbuilder.bat command.

When you start the Oracle Locale Builder, the screen shown in Figure 13–1 appears.

Figure 13–1 Oracle Locale Builder Utility

Oracle Locale Builder is a convenient tool for customizing locale data definitions. Use Oracle Locale Builder to view or create:

- Languages, including local month and day names, writing directions, etc.
- Territories, including calendar convention, date and time formats, number and monetary systems, etc.
- Character Sets, including character set type, character mappings and classifications, etc.
- Linguistic Sorts, including collation order, special collation rules, etc.

Oracle Locale Builder Windows and Dialog Boxes

Before using Oracle Locale Builder for a specific task, you should become familiar with tab pages and dialog boxes that include the following:
Overview of the Oracle Locale Builder Utility

- Existing Definitions Dialog Box
- Session Log Dialog Box
- Preview NLT Tab Page
- Open File Dialog Box

**Note:** Oracle Locale Builder includes online help.

### Existing Definitions Dialog Box

When you choose **New Language, New Territory, New Character Set, or New Linguistic Sort**, the first tab page that you see is labelled **General**. Click **Show Existing Definitions** to see the Existing Definitions dialog box.

The Existing Definitions dialog box enables you to open locale objects by name. If you know a specific language, territory, linguistic sort (collation), or character set that you want to start with, then click its displayed name. For example, you can open the AMERICAN language definition file as shown in Figure 13–2.

![Figure 13–2 Existing Definitions Dialog Box](image)

Choosing AMERICAN opens the 1x00001.nlb file. An NLB file is a binary file that contains the settings for a specific language, territory, character set, or linguistic sort.

Language and territory abbreviations are for reference only and cannot be opened.
Session Log Dialog Box

Choose **Tools > View Log** to see the Session Log dialog box. The Session Log dialog box shows what actions have been taken in the current session. Click **Save Log** to keep a record of all changes. **Figure 13–3** shows an example of a session log.

![Session Log Dialog Box](image)

**Figure 13–3  Session Log Dialog Box**

---

Preview NLT Tab Page

The NLT file is an XML file with the file extension `.nlt` that shows the settings for a specific language, territory, character set, or linguistic sort. The **Preview NLT** tab page presents a readable form of the file so that you can see whether the changes you have made look correct. You cannot modify the NLT file from the **Preview NLT** tab page. You must use the specific elements of the Oracle Locale Builder to modify the NLT file.

**Figure 13–4** shows an example of the **Preview NLT** tab page for a user-defined language called **AMERICAN FRENCH**.
Open File Dialog Box

You can see the Open File dialog box by choosing File > Open > By File Name. Then choose the NLB file that you want to modify or use as a template. An NLB file is a binary file with the file extension .nlb that contains the binary equivalent of the information in the NLT file. Figure 13–5 shows the Open File dialog box with the 1x00001.nlb file selected. The Preview pane shows that this NLB file is for the AMERICAN language.
Creating a New Language Definition with the Oracle Locale Builder

This section shows how to create a new language based on French. This new language is called AMERICAN FRENCH. First, open FRENCH from the Existing Definitions dialog box. Then change the language name to AMERICAN FRENCH and the Language Abbreviation to AF in the General tab page. Retain the default values for the other fields. Figure 13–6 shows the resulting General tab page.
The following restrictions apply when choosing names for locale objects such as languages:

- Names must contain only ASCII characters
- Names must start with a letter
- Language, territory, and character set names cannot contain underscores

The valid range for the Language ID field for a user-defined language is 1,000 to 10,000. You can accept the value provided by Oracle Locale Builder or you can specify a value within the range.
Note: Only certain ID ranges are valid values for user-defined LANGUAGE, TERRITORY, CHARACTER SET, MONOLINGUAL COLLATION, and MULTILINGUAL COLLATION definitions. The ranges are specified in the sections of this chapter that concern each type of user-defined locale object.

Figure 13–7 shows how to set month names using the Month Names tab page.

Figure 13–7  Month Names Tab Page

All names are shown as they appear in the NLT file. If you choose Yes for capitalization, then the month names are capitalized in your application, but they do not appear capitalized in the Month Names tab page.
Figure 13–8 shows the **Day Names** tab page.

You can choose day names for your user-defined language. All names are shown as they appear in the NLT file. If you choose **Yes** for capitalization, then the day names are capitalized in your application, but they do not appear capitalized in the **Day Names** tab page.

### Creating a New Territory Definition with the Oracle Locale Builder

This section shows how to create a new territory called **REDWOOD SHORES** and use **RS** as a territory abbreviation. The new territory is not based on an existing territory definition.
The basic tasks are as follows:

- Assign a territory name
- Choose formats for the calendar, numbers, date and time, and currency

*Figure 13–9* shows the *General* tab page with *REDWOOD SHORES* specified as the *Territory Name*, 1001 specified as the *Territory ID*, and *RS* specified as the *Territory Abbreviation*.

*Figure 13–9  General Tab Page for Territories*

![General Tab Page for Territories](image)

The valid range for *Territory ID* for a user-defined territory is 1000 to 10000.

*Figure 13–10* shows settings for calendar formats in the *Calendar* tab page.
Tuesday is set as the first day of the week, and the first week of the calendar year is set as an ISO week. The screen displays a sample calendar.

See Also:

- "Calendar Formats" on page 3-27 for more information about choosing the first day of the week and the first week of the calendar year
- "Customizing Calendars with the NLS Calendar Utility" on page 13-17 for information about customizing calendars themselves

Figure 13–11 shows the Date&Time tab page.
When you choose a format from a list, Oracle Locale Builder displays an example of the format. In this case, the Short Date Format is set to DD-MM-YY. The Short Time Format is set to HH24:MI:SS. The Oracle Date Format is set to DD-MM-YY. The Long Date Format is set to fmDay, Month dd, yyyy. The TimeStamp Timezone Format is not set.

You can also enter your own formats instead of using the selection from the drop-down menus.

See Also:
- "Date Formats" on page 3-20
- "Time Formats" on page 3-24
- "Customizing Time Zone Data" on page 13-17
Figure 13–12 shows the **Number** tab page.

**Figure 13–12 Choosing Number Formats**

A period has been chosen for the **Decimal Symbol**. The **Negative Sign Location** is specified to be on the left of the number. The **Numeric Group Separator** is a comma. The **Number Grouping** is specified as 3 digits. The **List Separator** is a comma. The **Measurement System** is metric. The **Rounding Indicator** is 4.

You can enter your own values instead of using values in the lists.

When you choose a format from a list, Oracle Locale Builder displays an example of the format.

**See Also:** "Numeric Formats" on page 3-31
Figure 13–13 shows settings for currency formats in the Monetary tab page.

**Figure 13–13  Choosing Currency Formats**

The Local Currency Symbol is set to $. The Alternative Currency Symbol is the euro symbol. The Currency Presentation shows one of several possible sequences of the local currency symbol, the debit symbol, and the number. The Decimal Symbol is the period. The Group Separator is the comma. The Monetary Number Grouping is 3. The Monetary Precision, or number of digits after the decimal symbol, is 3. The Credit Symbol is +. The Debit Symbol is -. The International Currency Separator is a blank space, so it is not visible in the field. The International Currency Symbol (ISO currency symbol) is USD. Oracle Locale Builder displays examples of the currency formats you have selected.

You can enter your own values instead of using the lists.
The rest of this section contains the following topics:

- Customizing Time Zone Data
- Customizing Calendars with the NLS Calendar Utility

Customizing Time Zone Data

The time zone files contain the valid time zone names. The following information is included for each time zone:

- Offset from Coordinated Universal Time (UTC)
- Transition times for daylight savings time
- Abbreviations for standard time and daylight savings time. The abbreviations are used with the time zone names.

Two time zone files are included in the Oracle home directory. The default file is `oracore/zoneinfo/timezone.dat`. More time zones are included in `oracore/zoneinfo/timezlrq.dat`.

See Also: "Choosing a Time Zone File" on page 4-20 for more information about the contents of the time zone files and how to install the larger time zone file.

Customizing Calendars with the NLS Calendar Utility

Oracle supports several calendars. All of them are defined with data derived from Oracle’s globalization support, but some of them may require the addition of ruler eras or deviation days in the future. To add this information without waiting for a new release of the Oracle database server, you can use an external file that is automatically loaded when the calendar functions are executed.

Calendar data is first defined in a text file. The text definition file must be converted into binary format. You can use the NLS Calendar Utility (`lxegen`) to convert the text definition file into binary format.

The name of the text definition file and its location are hard-coded and depend on the platform. On UNIX platforms, the file name is `lxecal.nlt`. It is located in the `$ORACLE_HOME/nls/demo` directory. A sample text definition file is included in the directory.

See Also: "Currency Formats" on page 3-34
The `lxegen` utility produces a binary file from the text definition file. The name of the binary file is also hard-coded and depends on the platform. On UNIX platforms, the name of the binary file is `lxecal.nlb`. The binary file is generated in the same directory as the text file and overwrites an existing binary file.

After the binary file has been generated, it is automatically loaded during system initialization. Do not move or rename the file.

Invoke the calendar utility from the command line as follows:

```
% lxegen
```

See Also:
- Operating system documentation for the location of the files on your system
- "Calendar Systems" on page A-28

**Displaying a Code Chart with the Oracle Locale Builder**

You can display and print the code charts of character sets with the Oracle Locale Builder. From the opening screen for Oracle Locale Builder, choose **File > New > Character Set**. **Figure 13–14** shows the resulting screen.
Click **Show Existing Definitions**. Highlight the character set you wish to display. **Figure 13–15** shows the Existing Definitions combo box with US7ASCII highlighted.

**Figure 13–15 Choosing US7ASCII in the Existing Definitions Dialog Box**
Click **Open** to choose the character set. **Figure 13–16** shows the General tab page when US7ASCII has been chosen.

**Figure 13–16  General Tab Page When US7ASCII Has Been Chosen**

Click the **Character Data Mapping** tab. **Figure 13–17** shows the Character Data Mapping tab page for US7ASCII.
Figure 13–17  Character Data Mapping Tab Page for US7ASCII

Click View CodeChart. Figure 13–18 shows the code chart for US7ASCII.
Creating a New Character Set Definition with the Oracle Locale Builder

Creating a New Character Set Definition with the Oracle Locale Builder

You can customize a character set to meet specific user needs. You can extend an existing encoded character set definition. User-defined characters are often used to encode special characters that represent the following:

- Proper names
- Historical Han characters that are not defined in an existing character set standard

It shows the encoded value of each character in the local character set, the glyph associated with each character, and the Unicode value of each character in the local character set.

If you want to print the code chart, then click **Print Page**.
Vendor-specific characters
- New symbols or characters that you define

This section describes how Oracle supports user-defined characters. It includes the following topics:
- Character Sets with User-Defined Characters
- Oracle Character Set Conversion Architecture
- Unicode 3.2 Private Use Area
- User-Defined Character Cross-References Between Character Sets
- Guidelines for Creating a New Character Set from an Existing Character Set
- Example: Creating a New Character Set Definition with the Oracle Locale Builder

**Character Sets with User-Defined Characters**

User-defined characters are typically supported within East Asian character sets. These East Asian character sets have at least one range of reserved code points for user-defined characters. For example, Japanese Shift-JIS preserves 1880 code points for user-defined characters. They are shown in Table 13–1.

<table>
<thead>
<tr>
<th>Japanese Shift JIS User-Defined Character Range</th>
<th>Number of Code Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>F040-F07E, F080-F0FC</td>
<td>188</td>
</tr>
<tr>
<td>F140-F17E, F180-F1FC</td>
<td>188</td>
</tr>
<tr>
<td>F240-F27E, F280-F2FC</td>
<td>188</td>
</tr>
<tr>
<td>F340-F37E, F380-F3FC</td>
<td>188</td>
</tr>
<tr>
<td>F440-F47E, F480-F4FC</td>
<td>188</td>
</tr>
<tr>
<td>F540-F57E, F580-F5FC</td>
<td>188</td>
</tr>
<tr>
<td>FF640-F67E, F680-F6FC</td>
<td>188</td>
</tr>
<tr>
<td>F740-F77E, F780-F7FC</td>
<td>188</td>
</tr>
<tr>
<td>F840-F87E, F880-F8FC</td>
<td>188</td>
</tr>
<tr>
<td>F940-F97E, F980-F9FC</td>
<td>188</td>
</tr>
</tbody>
</table>
The Oracle character sets listed in Table 13–2 contain predefined ranges that support user-defined characters.

### Table 13–2 Oracle Character Sets with User-Defined Character Ranges

<table>
<thead>
<tr>
<th>Character Set Name</th>
<th>Number of Code Points Available for User-Defined Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA16DBCS</td>
<td>4370</td>
</tr>
<tr>
<td>JA16EBCDIC930</td>
<td>4370</td>
</tr>
<tr>
<td>JA16SJIS</td>
<td>1880</td>
</tr>
<tr>
<td>JA16SJISYEN</td>
<td>1880</td>
</tr>
<tr>
<td>KO16DBCS</td>
<td>1880</td>
</tr>
<tr>
<td>KO16MSWIN949</td>
<td>1880</td>
</tr>
<tr>
<td>ZHS16DBCS</td>
<td>1880</td>
</tr>
<tr>
<td>ZHS16GBK</td>
<td>2149</td>
</tr>
<tr>
<td>ZHT16DBCS</td>
<td>6204</td>
</tr>
<tr>
<td>ZHT16MSWIN950</td>
<td>6217</td>
</tr>
</tbody>
</table>

### Oracle Character Set Conversion Architecture

The code point value that represents a particular character can vary among different character sets. A Japanese kanji character is shown in Figure 13–19.

![Japanese Kanji Character](image-url)

The following table shows how the character is encoded in different character sets.

<table>
<thead>
<tr>
<th>Unicode Encoding</th>
<th>JA16SJIS Encoding</th>
<th>JA16EUC Encoding</th>
<th>JA16DBCS Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>4E9C</td>
<td>889F</td>
<td>80A1</td>
<td>4867</td>
</tr>
</tbody>
</table>

In Oracle, all character sets are defined in terms of Unicode 3.2 code points. That is, each character is defined as a Unicode 3.2 code value. Character conversion takes place transparently to users by using Unicode as the intermediate form. For
example, when a JA16SJIS client connects to a JA16EUC database, the character shown in Figure 13–19 has the code point value 889F when it is entered from the JA16SJIS client. It is internally converted to Unicode (with code point value 4E9C) and then converted to JA16EUC (code point value B0A1).

**Unicode 3.2 Private Use Area**

Unicode 3.2 reserves the range E000-F8FF for the Private Use Area (PUA). The PUA is intended for private use character definition by end users or vendors.

User-defined characters can be converted between two Oracle character sets by using Unicode 3.2 PUA as the intermediate form, the same as standard characters.

**User-Defined Character Cross-References Between Character Sets**

Cross-references between different character sets are required when registering user-defined characters across operating systems. Cross-references ensure that the user-defined characters can be converted correctly across the different character sets.

For example, when registering a user-defined character on both a Japanese Shift-JIS operating system and a Japanese IBM Host operating system, you may want to assign the F040 code point on the Shift-JIS operating system and the 6941 code point on the IBM Host operating system for this character so that Oracle can map this character correctly between the character sets JA16SJIS and JA16DBCS.

User-defined character cross-reference information can be found by viewing the character set definitions using the Oracle Locale Builder. For example, you can determine that both the Shift-JIS UDC value F040 and the IBM Host UDC value 6941 are mapped to the same Unicode PUA value E000.

*See Also:* Appendix B, "Unicode Character Code Assignments"

**Guidelines for Creating a New Character Set from an Existing Character Set**

By default, the Oracle Locale Builder generates the next available character set ID for you. You can also choose your own character set ID. Use the following format for naming character set definition NLT files:

```
1x2dddd.nlt
```

*ddddd* is the 4-digit character set ID in hex.

When you modify a character set, observe the following guidelines:
Do not remap existing characters.

All character mappings must be unique.

New characters should be mapped into the Unicode private use range e000 to f4ff. (Note that the actual Unicode 3.2 private use range is e000-f8ff. However, Oracle reserves f500-f8ff for its own private use.)

No line in the character set definition file can be longer than 80 characters.

---

**Note:** When you create a new multibyte character set from an existing character set, use an 8-bit or multibyte character set as the original character set.

---

If you derive a new character set from an existing Oracle character set, then Oracle Corporation recommends using the following character set naming convention:

\(<\text{Oracle_character_set_name}\>\text{<organization_name>}\text{EXT<version>}

For example, if a company such as Sun Microsystems adds user-defined characters to the JA16EUC character set, then the following character set name is appropriate:

\(\text{JA16EUCSUNWEXT1}\)

The character set name contains the following parts:

- \(\text{JA16EUC}\) is the character set name defined by Oracle
- \(\text{SUNW}\) represents the organization name (company stock trading abbreviation for Sun Microsystems)
- \(\text{EXT}\) specifies that this character set is an extension to the JA16EUC character set
- \(\text{1}\) specifies the version

---

**Example: Creating a New Character Set Definition with the Oracle Locale Builder**

This section shows how to create a new character set called MYCHARSET with 10001 for its Character Set ID. The example uses the WE8ISO8859P1 character set and adds 10 Chinese characters.

**Figure 13–20** shows the General tab page for MYCHARSET.
Click **Show Existing Definitions** and choose the WE8ISO8859P1 character set from the Existing Definitions dialog box.

The **ISO Character Set ID** and **Base Character Set ID** fields are optional. The **Base Character Set ID** is used for inheriting values so that the properties of the base character set are used as a template. The **Character Set ID** is automatically generated, but you can override it. The valid range for a user-defined character set ID is 8000 to 8999 or 10000 to 20000.

**Note:** If you are using Pro*COBOL, then choose a character set ID between 8000 and 8999.

The **ISO Character Set ID** field remains blank for user-defined character sets.
Figure 13–21 shows the **Type Specification** tab page.

**Figure 13–21  Type Specification Tab Page**

The **Character Set Category** is **ASCII_BASED**. The **BYTE_UNIQUE** button is checked.

When you have chosen an existing character set, the fields for the **Type Specification** tab page should already be set to appropriate values. You should keep these values unless you have a specific reason for changing them. If you need to change the settings, then use the following guidelines:

- **FIXED_WIDTH** is used to identify character sets whose characters have a uniform length.
- **BYTE_UNIQUE** means that the single-byte range of code points is distinct from the multibyte range. The code in the first byte indicates whether the character is single-byte or multibyte. An example is JA16EUC.

- **DISPLAY** identifies character sets that are used only for display on clients and not for storage. Some Arabic, Devanagari, and Hebrew character sets are display character sets.

- **SHIFT** is used for character sets that require extra shift characters to distinguish between single-byte characters and multibyte characters.

  See Also:  "Variable-width multibyte encoding schemes" on page 2-10 for more information about shift-in and shift-out character sets

Figure 13–22 shows how to add user-defined characters.
Open the **Character Data Mapping** tab page. Highlight the character that you want to add characters after in the character set. In this example, the `0xff` local character value is highlighted.

You can add one character at a time or use a text file to import a large number of characters. In this example, a text file is imported. The first column is the local character value. The second column is the Unicode value. The file contains the following character values:

- 88a2 963f
- 88a3 54c0
- 88a4 611b
- 88a5 6328
- 88a6 59f6
- 88a7 9022
Creating a New Character Set Definition with the Oracle Locale Builder

Choose File > Import > User-Defined Characters Data.

Figure 13–23 shows that the imported characters are added after 0xff in the character set.

Figure 13–23  New Characters in the Character Set
Creating a New Linguistic Sort with the Oracle Locale Builder

This section shows how to create a new multilingual linguistic sort called \texttt{MY\_GENERIC\_M} with a collation ID of \texttt{10001}. The \texttt{GENERIC\_M} linguistic sort is used as the basis for the new linguistic sort. \texttt{Figure 13–24} shows how to begin.

\textbf{Figure 13–24} General Tab Page for Collation

Settings for the flags are automatically derived. \texttt{SWAP\_WITH\_NEXT} is relevant for Thai and Lao sorts. \texttt{REVERSE\_SECONDARY} is for French sorts. \texttt{CANONICAL\_EQUIVALENCE} determines whether canonical rules are used. In this example, \texttt{CANONICAL\_EQUIVALENCE} is checked.

The valid range for \texttt{Collation ID} (sort ID) for a user-defined sort is 1000 to 2000 for monolingual collation and 10000 to 11000 for multilingual collation.
Creating a New Linguistic Sort with the Oracle Locale Builder

See Also:

- Figure 13–28, "Canonical Rules Dialog Box" for more information about canonical rules
- Chapter 5, "Linguistic Sorting and String Searching"

Figure 13–25 shows the Unicode Collation Sequence tab page.

Figure 13–25  Unicode Collation Sequence Tab Page

This example customizes the linguistic sort by moving digits so that they sort after letters. Complete the following steps:
1. Highlight the Unicode value that you want to move. In Figure 13–25, the \x0034 Unicode value is highlighted. Its location in the Unicode Collation Sequence is called a node.

2. Click Cut. Select the location where you want to move the node.

3. Click Paste. Clicking Paste opens the Paste Node dialog box, shown in Figure 13–26.

4. The Paste Node dialog box enables you to choose whether to paste the node after or before the location you have selected. It also enables you to choose the level (Primary, Secondary, or Tertiary) of the node in relation to the node that you want to paste it next to.

   Select the position and the level at which you want to paste the node.

   In Figure 13–26, the After button and the Primary button are selected.

5. Click OK to paste the node.

Use similar steps to move other digits to a position after the letters a through z.

Figure 13–27 shows the resulting Unicode Collation Sequence tab page after the digits 0 through 4 have been moved to a position after the letters a through z.
The rest of this section contains the following topics:

- Changing the Sort Order for All Characters with the Same Diacritic
- Changing the Sort Order for One Character with a Diacritic

### Changing the Sort Order for All Characters with the Same Diacritic

This example shows how to change the sort order for characters with diacritics. You can do this by changing the sort for all characters containing a particular diacritic or by changing one character at a time. This example changes the sort of each character with a circumflex (for example, û) to be after the same character containing a tilde.
Verify the current sort order by choosing **Tools > Canonical Rules.** This opens the Canonical Rules dialog box, shown in Figure 13–28.

*Figure 13–28 Canonical Rules Dialog Box*

![Canonical Rules Dialog Box](image)

Figure 13–28 shows how characters are decomposed into their canonical equivalents and their current sorting orders. For example, ū is represented as u plus ^.

See Also: Chapter 5, "Linguistic Sorting and String Searching" for more information about canonical rules.

In the Oracle Locale Builder collation window (shown in Figure 13–24), click the **Non-Spacing Characters** tab. If you use the **Non-Spacing Characters** tab page, then changes for diacritics apply to all characters. Figure 13–29 shows the **Non-Spacing Characters** tab page.
Select the circumflex and click Cut. Click Yes in the Removal Confirmation dialog box. Select the tilde and click Paste. Choose After and Secondary in the Paste Node dialog box and click OK.

Figure 13–30 illustrates the new sort order.
Changing the Sort Order for One Character with a Diacritic

To change the order of a specific character with a diacritic, insert the character directly into the appropriate position. Characters with diacritics do not appear in the Unicode Collation Sequence tab page, so you cannot cut and paste them into the new location.

This example changes the sort order for ä so that it sorts after Z.

Select the Unicode Collation tab. Highlight the character, Z, that you want to put ä next to. Click Add. The Insert New Node dialog box appears, as shown in Figure 13–31.
Choose **After** and **Primary** in the Insert New Node dialog box. Enter the Unicode code point value of \( \ddot{a} \). The code point value is \( \text{x}00e4 \). Click **OK**.

**Figure 13–32** shows the resulting sort order.
Generating and Installing NLB Files

After you have defined a new language, territory, character set, or linguistic sort, generate new NLB files from the NLT files.

1. As the user who owns the files (typically user oracle), back up the NLS installation boot file (lx0boot.nl) and the NLS system boot file (lx1boot.nl) in the ORA_NLS10 directory. On a UNIX platform, enter commands similar to the following:

   % setenv ORA_NLS10 $ORACLE_HOME/nls/data
   % cd $ORA_NLS10
   % cp -p lx0boot.nl lx0boot.nl.orig
   % cp -p lx1boot.nl lx1boot.nl.orig
Note that the -p option preserves the timestamp of the original file.

2. In Oracle Locale Builder, choose Tools > Generate NLB or click the Generate NLB icon in the left side bar.

3. Click Browse to find the directory where the NLT file is located. The location dialog box is shown in Figure 13–33.

**Figure 13–33  Location Dialog Box**

![Location Dialog Box](image)

Do not try to specify an NLT file. Oracle Locale Builder generates an NLB file for each NLT file.

4. Click OK to generate the NLB files.

**Figure 13–34** illustrates the final notification that you have successfully generated NLB files for all NLT files in the directory.

**Figure 13–34  NLB Generation Success Dialog Box**

![NLB Generation Success Dialog Box](image)

5. Copy the `lx1boot.nlb` file into the path that is specified by the `ORA_NLS10` environment variable. For example, on a UNIX platform, enter a command similar to the following:

   ```
   % cp /directory_name/lx1boot.nlb $ORA_NLS10/lx1boot.nlb
   ```
6. Copy the new NLB files into the ORA_NLS10 directory. For example, on a UNIX platform, enter commands similar to the following:

```
% cp /directory_name/1x22710.nlb $ORA_NLS10
% cp /directory_name/1x52710.nlb $ORA_NLS10
```

**Note:** Oracle Locale Builder generates NLB files in the directory where the NLT files reside.

7. Restart the database to use the newly created locale data.

8. To use the new locale data on the client side, exit the client and re-invoke the client after installing the NLB files.

**See Also:** "Locale Data on Demand" on page 1-2 for more information about the ORA_NLS10 environment variable

---

**Transportable NLB Data**

NLB files that are generated on one platform can be transported to another platform by, for example, FTP. The transported NLB files can be used the same way as the NLB files that were generated on the original platform. This is convenient because locale data can be modified on one platform and copied to other platforms. Note that you must copy all of the NLB files from one platform to another, not just the files that have been modified. Also note that "Generating and Installing NLB Files" on page 13-40 is performed the same way as in previous releases. NLB files that are generated on a Solaris platform can be copied by FTP to a Windows platform and provide the same functionality there.

Different binary formats (such as 32-bit, 64-bit, big-endian, little-endian, ASCII, and EBCDIC) are processed during NLB loading in a manner that is transparent to the user.
This appendix lists the languages, territories, character sets, and other locale data supported by the Oracle server. It includes these topics:

- Languages
- Translated Messages
- Territories
- Character Sets
- Language and Character Set Detection Support
- Linguistic Sorts
- Calendar Systems
- Time Zone Names
- Obsolete Locale Data

You can obtain information about character sets, languages, territories, and sorting orders by querying the `V$NLS_VALID_VALUES` dynamic performance view.

**See Also:** Oracle Database Reference for more information about the data that can be returned by this view
Languages

Languages in Table A–1 provide support for locale-sensitive information such as the following:

- Day and month names and their abbreviations
- Symbols for equivalent expressions for A.M., P.M., A.D., and B.C.
- Default sorting sequence for character data when the `ORDER BY` SQL clause is specified
- Writing direction (left to right or right to left)
- Affirmative and negative response strings (for example, YES and NO)

By using Unicode databases and datatypes, you can store, process, and retrieve data for almost all contemporary languages, including many that do not appear in Table A–1.

<table>
<thead>
<tr>
<th>Language Name</th>
<th>Language Abbreviation</th>
<th>Default Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMERICAN</td>
<td>us</td>
<td>binary</td>
</tr>
<tr>
<td>ARABIC</td>
<td>ar</td>
<td>ARABIC</td>
</tr>
<tr>
<td>ASSAMESE</td>
<td>as</td>
<td>binary</td>
</tr>
<tr>
<td>AZERBAIJANI</td>
<td>az</td>
<td>AZERBAIJANI</td>
</tr>
<tr>
<td>BANGLA</td>
<td>bn</td>
<td>binary</td>
</tr>
<tr>
<td>BRAZILIAN PORTUGUESE</td>
<td>ptb</td>
<td>WEST_EUROPEAN</td>
</tr>
<tr>
<td>BULGARIAN</td>
<td>bg</td>
<td>BULGARIAN</td>
</tr>
<tr>
<td>CANADIAN FRENCH</td>
<td>frc</td>
<td>CANADIAN FRENCH</td>
</tr>
<tr>
<td>CATALAN</td>
<td>ca</td>
<td>CATALAN</td>
</tr>
<tr>
<td>CROATIAN</td>
<td>hr</td>
<td>CROATIAN</td>
</tr>
<tr>
<td>CYRILLIC KAZAKH</td>
<td>ckk</td>
<td>GENERIC_M</td>
</tr>
<tr>
<td>CYRILLIC SERBIAN</td>
<td>csr</td>
<td>GENERIC_M</td>
</tr>
<tr>
<td>CYRILLIC UZBEK</td>
<td>cuz</td>
<td>GENERIC_M</td>
</tr>
<tr>
<td>CZECH</td>
<td>cs</td>
<td>CZECH</td>
</tr>
<tr>
<td>DANISH</td>
<td>dk</td>
<td>DANISH</td>
</tr>
<tr>
<td>Language Name</td>
<td>Language Abbreviation</td>
<td>Default Sort</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>DUTCH</td>
<td>nl</td>
<td>DUTCH</td>
</tr>
<tr>
<td>EGYPTIAN</td>
<td>eg</td>
<td>ARABIC</td>
</tr>
<tr>
<td>ENGLISH</td>
<td>gb</td>
<td>binary</td>
</tr>
<tr>
<td>ESTONIAN</td>
<td>et</td>
<td>ESTONIAN</td>
</tr>
<tr>
<td>FINNISH</td>
<td>sf</td>
<td>FINNISH</td>
</tr>
<tr>
<td>FRENCH</td>
<td>f</td>
<td>FRENCH</td>
</tr>
<tr>
<td>GERMAN DIN</td>
<td>din</td>
<td>GERMAN</td>
</tr>
<tr>
<td>GERMAN</td>
<td>d</td>
<td>GERMAN</td>
</tr>
<tr>
<td>GREEK</td>
<td>el</td>
<td>GREEK</td>
</tr>
<tr>
<td>GUJARATI</td>
<td>gu</td>
<td>binary</td>
</tr>
<tr>
<td>HEBREW</td>
<td>iw</td>
<td>HEBREW</td>
</tr>
<tr>
<td>HINDI</td>
<td>hi</td>
<td>binary</td>
</tr>
<tr>
<td>HUNGARIAN</td>
<td>hu</td>
<td>HUNGARIAN</td>
</tr>
<tr>
<td>ICELANDIC</td>
<td>is</td>
<td>ICELANDIC</td>
</tr>
<tr>
<td>INDONESIAN</td>
<td>in</td>
<td>INDONESIAN</td>
</tr>
<tr>
<td>ITALIAN</td>
<td>i</td>
<td>WEST_EUROPEAN</td>
</tr>
<tr>
<td>JAPANESE</td>
<td>ja</td>
<td>binary</td>
</tr>
<tr>
<td>KANNADA</td>
<td>kn</td>
<td>binary</td>
</tr>
<tr>
<td>KOREAN</td>
<td>ko</td>
<td>binary</td>
</tr>
<tr>
<td>LATIN AMERICAN SPANISH</td>
<td>esa</td>
<td>SPANISH</td>
</tr>
<tr>
<td>LATIN SERBIAN</td>
<td>lsr</td>
<td>binary</td>
</tr>
<tr>
<td>LATIN UZBEK</td>
<td>luz</td>
<td>GENERIC_M</td>
</tr>
<tr>
<td>LATVIAN</td>
<td>lv</td>
<td>LATVIAN</td>
</tr>
<tr>
<td>LITHUANIAN</td>
<td>lt</td>
<td>LITHUANIAN</td>
</tr>
<tr>
<td>MACEDONIAN</td>
<td>mk</td>
<td>binary</td>
</tr>
<tr>
<td>MALAY</td>
<td>ms</td>
<td>MALAY</td>
</tr>
</tbody>
</table>
## Table A–1  Oracle Supported Languages (Cont.)

<table>
<thead>
<tr>
<th>Language Name</th>
<th>Language Abbreviation</th>
<th>Default Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>MALAYALAM</td>
<td>ml</td>
<td>binary</td>
</tr>
<tr>
<td>MARATHI</td>
<td>mr</td>
<td>binary</td>
</tr>
<tr>
<td>MEXICAN SPANISH</td>
<td>esm</td>
<td>WEST_EUROPEAN</td>
</tr>
<tr>
<td>NORWEGIAN</td>
<td>n</td>
<td>NORWEGIAN</td>
</tr>
<tr>
<td>ORIYA</td>
<td>or</td>
<td>binary</td>
</tr>
<tr>
<td>POLISH</td>
<td>pl</td>
<td>POLISH</td>
</tr>
<tr>
<td>PORTUGUESE</td>
<td>pt</td>
<td>WEST_EUROPEAN</td>
</tr>
<tr>
<td>PUNJABI</td>
<td>pa</td>
<td>binary</td>
</tr>
<tr>
<td>ROMANIAN</td>
<td>ro</td>
<td>ROMANIAN</td>
</tr>
<tr>
<td>RUSSIAN</td>
<td>ru</td>
<td>RUSSIAN</td>
</tr>
<tr>
<td>SIMPLIFIED CHINESE</td>
<td>zhs</td>
<td>binary</td>
</tr>
<tr>
<td>SLOVAK</td>
<td>sk</td>
<td>SLOVAK</td>
</tr>
<tr>
<td>SLOVENIAN</td>
<td>sl</td>
<td>SLOVENIAN</td>
</tr>
<tr>
<td>SPANISH</td>
<td>e</td>
<td>SPANISH</td>
</tr>
<tr>
<td>SWEDISH</td>
<td>s</td>
<td>SWEDISH</td>
</tr>
<tr>
<td>TAMIL</td>
<td>ta</td>
<td>binary</td>
</tr>
<tr>
<td>TELUGU</td>
<td>te</td>
<td>binary</td>
</tr>
<tr>
<td>THAI</td>
<td>th</td>
<td>THAI_DICTIONARY</td>
</tr>
<tr>
<td>TRADITIONAL CHINESE</td>
<td>zht</td>
<td>binary</td>
</tr>
<tr>
<td>TURKISH</td>
<td>tr</td>
<td>TURKISH</td>
</tr>
<tr>
<td>UKRAINIAN</td>
<td>uk</td>
<td>UKRAINIAN</td>
</tr>
<tr>
<td>VIETNAMESE</td>
<td>vn</td>
<td>VIETNAMESE</td>
</tr>
</tbody>
</table>

## Translated Messages

Oracle error messages have been translated into the languages which are listed in Table A–2.
### Table A–2  Oracle Supported Messages

<table>
<thead>
<tr>
<th>Name</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARABIC</td>
<td>ar</td>
</tr>
<tr>
<td>BRAZILIAN PORTUGUESE</td>
<td>ptb</td>
</tr>
<tr>
<td>CATALAN</td>
<td>ca</td>
</tr>
<tr>
<td>CZECH</td>
<td>cs</td>
</tr>
<tr>
<td>DANISH</td>
<td>dk</td>
</tr>
<tr>
<td>DUTCH</td>
<td>nl</td>
</tr>
<tr>
<td>FINNISH</td>
<td>sf</td>
</tr>
<tr>
<td>FRENCH</td>
<td>f</td>
</tr>
<tr>
<td>GERMAN</td>
<td>d</td>
</tr>
<tr>
<td>GREEK</td>
<td>el</td>
</tr>
<tr>
<td>HEBREW</td>
<td>iw</td>
</tr>
<tr>
<td>HUNGARIAN</td>
<td>hu</td>
</tr>
<tr>
<td>ITALIAN</td>
<td>i</td>
</tr>
<tr>
<td>JAPANESE</td>
<td>ja</td>
</tr>
<tr>
<td>KOREAN</td>
<td>ko</td>
</tr>
<tr>
<td>NORWEGIAN</td>
<td>n</td>
</tr>
<tr>
<td>POLISH</td>
<td>pl</td>
</tr>
<tr>
<td>PORTUGUESE</td>
<td>pt</td>
</tr>
<tr>
<td>ROMANIAN</td>
<td>ro</td>
</tr>
<tr>
<td>RUSSIAN</td>
<td>ru</td>
</tr>
<tr>
<td>SIMPLIFIED CHINESE</td>
<td>zhs</td>
</tr>
<tr>
<td>SLOVAK</td>
<td>sk</td>
</tr>
<tr>
<td>SPANISH</td>
<td>e</td>
</tr>
<tr>
<td>SWEDISH</td>
<td>s</td>
</tr>
<tr>
<td>THAI</td>
<td>th</td>
</tr>
<tr>
<td>TRADITIONAL CHINESE</td>
<td>zht</td>
</tr>
<tr>
<td>TURKISH</td>
<td>tr</td>
</tr>
</tbody>
</table>
## Territories

Table A–3 lists the territories supported by the Oracle server.

### Table A–3 Oracle Supported Territories

<table>
<thead>
<tr>
<th>Name</th>
<th>Name</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGERIA</td>
<td>HONG KONG</td>
<td>PERU</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>PHILIPPINES</td>
</tr>
<tr>
<td>AMERICA</td>
<td>HUNGARY</td>
<td>POLAND</td>
</tr>
<tr>
<td>AUSTRALIA</td>
<td>ICELAND</td>
<td>PORTUGAL</td>
</tr>
<tr>
<td>AUSTRIA</td>
<td>INDIA</td>
<td>PUERTO RICO</td>
</tr>
<tr>
<td>BAHRAIN</td>
<td>INDONESIA</td>
<td>QATAR</td>
</tr>
<tr>
<td>BANGLADESH</td>
<td>IRAQ</td>
<td>ROMANIA</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>RUSSIA</td>
</tr>
<tr>
<td>BELGIUM</td>
<td>IRELAND</td>
<td>SAUDI ARABIA</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>SERBIA AND MONTENEGRO</td>
</tr>
<tr>
<td>BRAZIL</td>
<td>ISRAEL</td>
<td>SINGAPORE</td>
</tr>
<tr>
<td>BULGARIA</td>
<td>ITALY</td>
<td>SLOVAKIA</td>
</tr>
<tr>
<td>CANADA</td>
<td>JAPAN</td>
<td>SLOVENIA</td>
</tr>
<tr>
<td>CATALONIA</td>
<td>JORDAN</td>
<td>SOMALIA</td>
</tr>
<tr>
<td>CHILE</td>
<td>KAZAKHSTAN</td>
<td>SOUTH AFRICA</td>
</tr>
<tr>
<td>CHINA</td>
<td>KOREA</td>
<td>SPAIN</td>
</tr>
<tr>
<td>ARGENTINA</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>AZERBAIJAN</td>
<td>KUWAIT</td>
<td>SUDAN</td>
</tr>
<tr>
<td>COLOMBIA</td>
<td>LATVIA</td>
<td>SWEDEN</td>
</tr>
<tr>
<td>COSTA RICA</td>
<td>LEBANON</td>
<td>SWITZERLAND</td>
</tr>
<tr>
<td>CROATIA</td>
<td>LIBYA</td>
<td>SYRIA</td>
</tr>
<tr>
<td>CYPRUS</td>
<td>LITHUANIA</td>
<td>TAIWAN</td>
</tr>
<tr>
<td>CZECH REPUBLIC</td>
<td>LUXEMBOURG</td>
<td>THAILAND</td>
</tr>
<tr>
<td>DENMARK</td>
<td>-</td>
<td>THE NETHERLANDS</td>
</tr>
</tbody>
</table>
Oracle-supported character sets are listed in the following sections according to three broad language groups.

- **Asian Language Character Sets**
- **European Language Character Sets**
- **Middle Eastern Language Character Sets**

In addition, common subset/superset combinations are listed.

Note that some character sets may be listed under multiple language groups because they provide multilingual support. For instance, Unicode spans the Asian, European, and Middle Eastern language groups because it supports most of the major scripts of the world.

The comment section indicates the type of encoding used:

- SB = Single-byte encoding
- MB = Multibyte encoding
- FIXED = Fixed-width multibyte encoding
As mentioned in Chapter 3, "Setting Up a Globalization Support Environment", the type of encoding affects performance, so use the most efficient encoding that meets your language needs. Also, some encoding types can only be used with certain data types. For instance, the AL16UTF16 character set can only be used as an NCHAR character set, and not as a database character set.

Also documented in the comment section are other unique features of the character set that may be important to users or your database administrator. For example, the information includes whether the character set supports the euro currency symbol, whether user-defined characters are supported, and whether the character set is a strict superset of ASCII. (You can use the `ALTER DATABASE CHARACTER SET` statement to migrate from the US7ASCII character set to a character set that is a strict superset of ASCII.)

The following is the key for the comment column of the character set tables:

- **EURO**: euro symbol supported
- **UDC**: user-defined characters supported
- **ASCII**: strict superset of ASCII
- **MB**: multibyte
- **SB**: single-byte

Oracle does not document individual code page layouts. For specific details about a particular character set, its character repertoire, and code point values, you can use Oracle Locale Builder may be used. Otherwise, you should refer to the actual national, international, or vendor-specific standards.

**See Also:** Chapter 13, "Customizing Locale"

### Asian Language Character Sets

Table A–4 lists the Oracle character sets that can support Asian languages. The list is alphabetical by description.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN8BSCII</td>
<td>Bangladesh National Code 8-bit BSCII</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>ZHT16BIG5</td>
<td>BIG5 16-bit Traditional Chinese</td>
<td>MB, ASCII</td>
</tr>
<tr>
<td>ZHS16CGB231280</td>
<td>CGB2312-80 16-bit Simplified Chinese</td>
<td>MB, ASCII</td>
</tr>
<tr>
<td>JA16EUC</td>
<td>EUC 24-bit Japanese</td>
<td>MB, ASCII</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Comments</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>JA16EUCTILDE</td>
<td>The same as JA16EUC except for the way that the wave dash and the tilde are mapped to and from Unicode.</td>
<td>MB, ASCII</td>
</tr>
<tr>
<td>JA16EUCYEN</td>
<td>EUC 24-bit Japanese with ' mapped to the Japanese yen character</td>
<td>MB</td>
</tr>
<tr>
<td>ZHT32EUC</td>
<td>EUC 32-bit Traditional Chinese</td>
<td>MB, ASCII</td>
</tr>
<tr>
<td>ZHS16GBK</td>
<td>GBK 16-bit Simplified Chinese</td>
<td>MB, ASCII, UDC</td>
</tr>
<tr>
<td>ZHS32GB18030</td>
<td>GB18030-2000</td>
<td>MB, ASCII, EURO</td>
</tr>
<tr>
<td>ZHT16CCDC</td>
<td>HP CCDC 16-bit Traditional Chinese</td>
<td>MB, ASCII</td>
</tr>
<tr>
<td>JA16DBCS</td>
<td>IBM EBCDIC 16-bit Japanese</td>
<td>MB, UDC</td>
</tr>
<tr>
<td>JA16EBCDIC930</td>
<td>IBM DBCS Code Page 290 16-bit Japanese</td>
<td>MB, UDC</td>
</tr>
<tr>
<td>KO16DBCS</td>
<td>IBM EBCDIC 16-bit Korean</td>
<td>MB, UDC</td>
</tr>
<tr>
<td>ZHS16DBCS</td>
<td>IBM EBCDIC 16-bit Simplified Chinese</td>
<td>MB, UDC</td>
</tr>
<tr>
<td>ZHT16DBCS</td>
<td>IBM EBCDIC 16-bit Traditional Chinese</td>
<td>MB, UDC</td>
</tr>
<tr>
<td>JA16VMS</td>
<td>JVMS 16-bit Japanese</td>
<td>MB, ASCII</td>
</tr>
<tr>
<td>KO16KSC5601</td>
<td>KSC5601 16-bit Korean</td>
<td>MB, ASCII</td>
</tr>
<tr>
<td>KO16KSCCS</td>
<td>KSCCS 16-bit Korean</td>
<td>MB, ASCII</td>
</tr>
<tr>
<td>ZHS16MACCGB231280</td>
<td>Mac client CGB2312-80 16-bit Simplified Chinese</td>
<td>MB</td>
</tr>
<tr>
<td>JA16MACSJIS</td>
<td>Mac client Shift-JIS 16-bit Japanese</td>
<td>MB</td>
</tr>
<tr>
<td>TH8MACTHAI</td>
<td>Mac Client 8-bit Latin/Thai</td>
<td>SB</td>
</tr>
<tr>
<td>TH8MACTHAIS</td>
<td>Mac Server 8-bit Latin/Thai</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>TH8TISEBCDICS</td>
<td>Thai Industrial Standard 620-2533-EBCDIC Server 8-bit</td>
<td>SB</td>
</tr>
<tr>
<td>KO16MSWIN949</td>
<td>MS Windows Code Page 949 Korean</td>
<td>MB, ASCII, UDC</td>
</tr>
<tr>
<td>ZHT16HKSCS</td>
<td>MS Windows Code Page 950 with Hong Kong Supplementary Character Set HKSCS-2001 (character set conversion to and from Unicode is based on Unicode 3.0)</td>
<td>MB, ASCII, EURO</td>
</tr>
<tr>
<td>ZHT16MSWIN950</td>
<td>MS Windows Code Page 950 Traditional Chinese</td>
<td>MB, ASCII, UDC</td>
</tr>
<tr>
<td>VN8MSWIN1258</td>
<td>MS Windows Code Page 1258 8-bit Vietnamese</td>
<td>SB, ASCII, EURO</td>
</tr>
<tr>
<td>IN8ISCII</td>
<td>Multiple-Script Indian Standard 8-bit Latin/Indian Languages</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>JA16SJIS</td>
<td>Shift-JIS 16-bit Japanese</td>
<td>MB, ASCII, UDC</td>
</tr>
</tbody>
</table>
Table A–4  Asian Language Character Sets (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA16SJISTILDE</td>
<td>The same as JA16SJIS except for the way that the wave dash and the tilde are mapped to and from Unicode.</td>
<td>MB, ASCII, UDC</td>
</tr>
<tr>
<td>JA16SJISYEN</td>
<td>Shift-JIS 16-bit Japanese with ‘\’ mapped to the Japanese yen character</td>
<td>MB, UDC</td>
</tr>
<tr>
<td>ZHT32SOPS</td>
<td>SOPS 32-bit Traditional Chinese</td>
<td>MB, ASCII</td>
</tr>
<tr>
<td>ZHT16DBT</td>
<td>Taiwan Taxation 16-bit Traditional Chinese</td>
<td>MB, ASCII</td>
</tr>
<tr>
<td>TH8TISASCII</td>
<td>Thai Industrial Standard 620-2533 - ASCII 8-bit</td>
<td>SB, ASCII, EURO</td>
</tr>
<tr>
<td>TH8TISEBCDIC</td>
<td>Thai Industrial Standard 620-2533 - EBCDIC 8-bit</td>
<td>SB</td>
</tr>
<tr>
<td>ZHT32TRIS</td>
<td>TRIS 32-bit Traditional Chinese</td>
<td>MB, ASCII</td>
</tr>
<tr>
<td>AL16UTF16</td>
<td>See “Universal Character Sets” on page A-19 for details</td>
<td>MB, EURO, FIXED</td>
</tr>
<tr>
<td>AL32UTF8</td>
<td>See “Universal Character Sets” on page A-19 for details</td>
<td>MB, ASCII, EURO</td>
</tr>
<tr>
<td>UTF8</td>
<td>See “Universal Character Sets” on page A-19 for details</td>
<td>MB, ASCII, EURO</td>
</tr>
<tr>
<td>UTF8E</td>
<td>See “Universal Character Sets” on page A-19 for details</td>
<td>MB, EURO</td>
</tr>
<tr>
<td>VN8VN3</td>
<td>VN3 8-bit Vietnamese</td>
<td>SB, ASCII</td>
</tr>
</tbody>
</table>

European Language Character Sets

Table A–5 lists the Oracle character sets that can support European languages. The list is alphabetical by description.

Table A–5  European Language Character Sets

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>US7ASCII</td>
<td>ASCII 7-bit American</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>SF7ASCII</td>
<td>ASCII 7-bit Finnish</td>
<td>SB</td>
</tr>
<tr>
<td>YUG7ASCII</td>
<td>ASCII 7-bit Yugoslavian</td>
<td>SB</td>
</tr>
<tr>
<td>RU8BESTA</td>
<td>BESTA 8-bit Latin/Cyrillic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>EL8GCOS7</td>
<td>Bull EBCDIC GCOS7 8-bit Greek</td>
<td>SB</td>
</tr>
<tr>
<td>WE8GCOS7</td>
<td>Bull EBCDIC GCOS7 8-bit West European</td>
<td>SB</td>
</tr>
<tr>
<td>EL8DEC</td>
<td>DEC 8-bit Latin/Greek</td>
<td>SB</td>
</tr>
<tr>
<td>TR7DEC</td>
<td>DEC VT100 7-bit Turkish</td>
<td>SB</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Comments</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>TR8DEC</td>
<td>DEC 8-bit Turkish</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>WE8DEC</td>
<td>DEC 8-bit West European</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>D7DEC</td>
<td>DEC VT100 7-bit German</td>
<td>SB</td>
</tr>
<tr>
<td>F7DEC</td>
<td>DEC VT100 7-bit French</td>
<td>SB</td>
</tr>
<tr>
<td>S7DEC</td>
<td>DEC VT100 7-bit Swedish</td>
<td>SB</td>
</tr>
<tr>
<td>E7DEC</td>
<td>DEC VT100 7-bit Spanish</td>
<td>SB</td>
</tr>
<tr>
<td>NDK7DEC</td>
<td>DEC VT100 7-bit Norwegian/Danish</td>
<td>SB</td>
</tr>
<tr>
<td>I7DEC</td>
<td>DEC VT100 7-bit Italian</td>
<td>SB</td>
</tr>
<tr>
<td>NL7DEC</td>
<td>DEC VT100 7-bit Dutch</td>
<td>SB</td>
</tr>
<tr>
<td>CH7DEC</td>
<td>DEC VT100 7-bit Swiss (German/French)</td>
<td>SB</td>
</tr>
<tr>
<td>SF7DEC</td>
<td>DEC VT100 7-bit Finnish</td>
<td>SB</td>
</tr>
<tr>
<td>WE8DG</td>
<td>DG 8-bit West European</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>WE8EBCDIC37C</td>
<td>EBCDIC Code Page 37 8-bit Oracle/c</td>
<td>SB</td>
</tr>
<tr>
<td>WE8EBCDIC37</td>
<td>EBCDIC Code Page 37 8-bit West European</td>
<td>SB</td>
</tr>
<tr>
<td>D8EBCDIC273</td>
<td>EBCDIC Code Page 273/1 8-bit Austrian German</td>
<td>SB</td>
</tr>
<tr>
<td>DK8EBCDIC277</td>
<td>EBCDIC Code Page 277/1 8-bit Danish</td>
<td>SB</td>
</tr>
<tr>
<td>S8EBCDIC278</td>
<td>EBCDIC Code Page 278/1 8-bit Swedish</td>
<td>SB</td>
</tr>
<tr>
<td>I8EBCDIC280</td>
<td>EBCDIC Code Page 280/1 8-bit Italian</td>
<td>SB</td>
</tr>
<tr>
<td>WE8EBCDIC284</td>
<td>EBCDIC Code Page 284 8-bit Latin American/Spanish</td>
<td>SB</td>
</tr>
<tr>
<td>WE8EBCDIC285</td>
<td>EBCDIC Code Page 285 8-bit West European</td>
<td>SB</td>
</tr>
<tr>
<td>WE8EBCDIC1047</td>
<td>EBCDIC Code Page 1047 8-bit West European</td>
<td>SB</td>
</tr>
<tr>
<td>WE8EBCDIC1140</td>
<td>EBCDIC Code Page 1140 8-bit West European</td>
<td>SB, EURO</td>
</tr>
<tr>
<td>WE8EBCDIC1140C</td>
<td>EBCDIC Code Page 1140 Client 8-bit West European</td>
<td>SB, EURO</td>
</tr>
<tr>
<td>WE8EBCDIC1145</td>
<td>EBCDIC Code Page 1145 8-bit West European</td>
<td>SB, EURO</td>
</tr>
<tr>
<td>WE8EBCDIC1146</td>
<td>EBCDIC Code Page 1146 8-bit West European</td>
<td>SB, EURO</td>
</tr>
<tr>
<td>WE8EBCDIC1148</td>
<td>EBCDIC Code Page 1148 8-bit West European</td>
<td>SB, EURO</td>
</tr>
<tr>
<td>WE8EBCDIC1148C</td>
<td>EBCDIC Code Page 1148 Client 8-bit West European</td>
<td>SB, EURO</td>
</tr>
</tbody>
</table>

Table A–5  European Language Character Sets (Cont.)
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>F8EBCDIC297</td>
<td>EBCDIC Code Page 297 8-bit French</td>
<td>SB</td>
</tr>
<tr>
<td>WE8EBCDIC500C</td>
<td>EBCDIC Code Page 500 8-bit Oracle/c</td>
<td>SB</td>
</tr>
<tr>
<td>WE8EBCDIC500</td>
<td>EBCDIC Code Page 500 8-bit West European</td>
<td>SB</td>
</tr>
<tr>
<td>EE8EBCDIC870</td>
<td>EBCDIC Code Page 870 8-bit East European</td>
<td>SB</td>
</tr>
<tr>
<td>EE8EBCDIC870C</td>
<td>EBCDIC Code Page 870 Client 8-bit East European</td>
<td>SB</td>
</tr>
<tr>
<td>EE8EBCDIC870S</td>
<td>EBCDIC Code Page 870 Server 8-bit East European</td>
<td>SB</td>
</tr>
<tr>
<td>WE8EBCDIC871</td>
<td>EBCDIC Code Page 871 8-bit Icelandic</td>
<td>SB</td>
</tr>
<tr>
<td>EL8EBCDIC875</td>
<td>EBCDIC Code Page 875 8-bit Greek</td>
<td>SB</td>
</tr>
<tr>
<td>EL8EBCDIC875R</td>
<td>EBCDIC Code Page 875 Server 8-bit Greek</td>
<td>SB</td>
</tr>
<tr>
<td>CL8EBCDIC1025</td>
<td>EBCDIC Code Page 1025 8-bit Cyrillic</td>
<td>SB</td>
</tr>
<tr>
<td>CL8EBCDIC1025C</td>
<td>EBCDIC Code Page 1025 Client 8-bit Cyrillic</td>
<td>SB</td>
</tr>
<tr>
<td>CL8EBCDIC1025R</td>
<td>EBCDIC Code Page 1025 Server 8-bit Cyrillic</td>
<td>SB</td>
</tr>
<tr>
<td>CL8EBCDIC1025S</td>
<td>EBCDIC Code Page 1025 Server 8-bit Cyrillic</td>
<td>SB</td>
</tr>
<tr>
<td>CL8EBCDIC1025X</td>
<td>EBCDIC Code Page 1025 (Modified) 8-bit Cyrillic</td>
<td>SB</td>
</tr>
<tr>
<td>TR8EBCDIC1026</td>
<td>EBCDIC Code Page 1026 8-bit Turkish</td>
<td>SB</td>
</tr>
<tr>
<td>TR8EBCDIC1026S</td>
<td>EBCDIC Code Page 1026 Server 8-bit Turkish</td>
<td>SB</td>
</tr>
<tr>
<td>BLT8EBCDIC1112</td>
<td>EBCDIC Code Page 1112 8-bit Baltic Multilingual</td>
<td>SB</td>
</tr>
<tr>
<td>BLT8EBCDIC1112S</td>
<td>EBCDIC Code Page 1112 8-bit Server Baltic Multilingual</td>
<td>SB</td>
</tr>
<tr>
<td>D8EBCDIC1141</td>
<td>EBCDIC Code Page 1141 8-bit Austrian German</td>
<td>SB, EURO</td>
</tr>
<tr>
<td>DK8EBCDIC1142</td>
<td>EBCDIC Code Page 1142 8-bit Danish</td>
<td>SB, EURO</td>
</tr>
<tr>
<td>S8EBCDIC1143</td>
<td>EBCDIC Code Page 1143 8-bit Swedish</td>
<td>SB, EURO</td>
</tr>
<tr>
<td>I8EBCDIC1144</td>
<td>EBCDIC Code Page 1144 8-bit Italian</td>
<td>SB, EURO</td>
</tr>
<tr>
<td>F8EBCDIC1147</td>
<td>EBCDIC Code Page 1147 8-bit French</td>
<td>SB, EURO</td>
</tr>
<tr>
<td>CL8EBCDIC1158</td>
<td>EBCDIC Code Page 1158 8-bit Cyrillic</td>
<td>SB</td>
</tr>
<tr>
<td>CL8EBCDIC1158R</td>
<td>EBCDIC Code Page 1158 Server 8-bit Cyrillic</td>
<td>SB</td>
</tr>
<tr>
<td>EEC8EUROASCI</td>
<td>EEC Targon 35 ASCI West European/Greek</td>
<td>SB</td>
</tr>
<tr>
<td>EEC8EUROPA3</td>
<td>EEC EUROPA3 8-bit West European/Greek</td>
<td>SB</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Comments</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>LA8PASSPORT</td>
<td>German Government Printer 8-bit All-European Latin</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>WE8HP</td>
<td>HP LaserJet 8-bit West European</td>
<td>SB</td>
</tr>
<tr>
<td>WE8ROMAN8</td>
<td>HP Roman8 8-bit West European</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>HU8CW12</td>
<td>Hungarian 8-bit CWI-2</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>HU8ABMOD</td>
<td>Hungarian 8-bit Special AB Mod</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>EL8EBCDIC423R</td>
<td>IBM EBCDIC Code Page 423 for RDBMS server-side</td>
<td>SB</td>
</tr>
<tr>
<td>LV8RST104090</td>
<td>IBM-PC Alternative Code Page 8-bit Latvian (Latin/Cyrillic)</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>US8PC437</td>
<td>IBM-PC Code Page 437 8-bit American</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>BG8PC437S</td>
<td>IBM-PC Code Page 437 8-bit (Bulgarian Modification)</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>EL8PC437S</td>
<td>IBM-PC Code Page 437 8-bit (Greek modification)</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>EL8PC737</td>
<td>IBM-PC Code Page 737 8-bit Greek/Latin</td>
<td>SB</td>
</tr>
<tr>
<td>LT8PC772</td>
<td>IBM-PC Code Page 772 8-bit Lithuanian (Latin/Cyrillic)</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>LT8PC774</td>
<td>IBM-PC Code Page 774 8-bit Lithuanian (Latin)</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>BLT8PC775</td>
<td>IBM-PC Code Page 775 8-bit Baltic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>WE8PC850</td>
<td>IBM-PC Code Page 850 8-bit West European</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>EL8PC851</td>
<td>IBM-PC Code Page 851 8-bit Greek/Latin</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>EE8PC852</td>
<td>IBM-PC Code Page 852 8-bit East European</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>RU8PC855</td>
<td>IBM-PC Code Page 855 8-bit Latin/Cyrillic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>TR8PC857</td>
<td>IBM-PC Code Page 857 8-bit Turkish</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>WE8PC858</td>
<td>IBM-PC Code Page 858 8-bit West European</td>
<td>SB, ASCII, EURO</td>
</tr>
<tr>
<td>WE8PC860</td>
<td>IBM-PC Code Page 860 8-bit West European</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>IS8PC861</td>
<td>IBM-PC Code Page 861 8-bit Icelandic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>CDN8PC863</td>
<td>IBM-PC Code Page 863 8-bit Canadian French</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>N8PC865</td>
<td>IBM-PC Code Page 865 8-bit Norwegian</td>
<td>SB. ASCII</td>
</tr>
<tr>
<td>RU8PC866</td>
<td>IBM-PC Code Page 866 8-bit Latin/Cyrillic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>EL8PC869</td>
<td>IBM-PC Code Page 869 8-bit Greek/Latin</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>LV8PC1117</td>
<td>IBM-PC Code Page 1117 8-bit Latvian</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Comments</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>US8ICL</td>
<td>ICL EBCDIC 8-bit American</td>
<td>SB</td>
</tr>
<tr>
<td>WE8ICL</td>
<td>ICL EBCDIC 8-bit West European</td>
<td>SB</td>
</tr>
<tr>
<td>WE8ISOICLUK</td>
<td>ICL special version ISO8859-1</td>
<td>SB</td>
</tr>
<tr>
<td>WE8ISO8859P1</td>
<td>ISO 8859-1 West European</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>EE8ISO8859P2</td>
<td>ISO 8859-2 East European</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>SE8ISO8859P3</td>
<td>ISO 8859-3 South European</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>NEE8ISO8859P4</td>
<td>ISO 8859-4 North and North-East European</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>CL8ISO8859P5</td>
<td>ISO 8859-5 Latin/Cyrillic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>AR8ISO8859P6</td>
<td>ISO 8859-6 Latin/Arabic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>EL8ISO8859P7</td>
<td>ISO 8859-7 Latin/Greek</td>
<td>SB, ASCII, EURO</td>
</tr>
<tr>
<td>IW8ISO8859P8</td>
<td>ISO 8859-8 Latin/Hebrew</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>NE8ISO8859P10</td>
<td>ISO 8859-10 North European</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>BLT8ISO8859P13</td>
<td>ISO 8859-13 Baltic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>CEL8ISO8859P14</td>
<td>ISO 8859-13 Celtic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>WE8ISO8859P15</td>
<td>ISO 8859-15 West European</td>
<td>SB, ASCII, EURO</td>
</tr>
<tr>
<td>LA8ISO6937</td>
<td>ISO 6937 8-bit Coded Character Set for Text Communication</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>CL8ISOIR111</td>
<td>ISOIR111 Cyrillic</td>
<td>SB</td>
</tr>
<tr>
<td>IW7IS960</td>
<td>Israeli Standard 960 7-bit Latin/Hebrew</td>
<td>SB</td>
</tr>
<tr>
<td>CL8KOI8U</td>
<td>KOI8 Ukrainian Cyrillic</td>
<td>SB</td>
</tr>
<tr>
<td>WE8EBCDIC1047E</td>
<td>Latin 1/Open Systems 1047</td>
<td>SB, EBCDIC, EURO</td>
</tr>
<tr>
<td>WE8EBCDIC924</td>
<td>Latin 9 EBCDIC 924</td>
<td>SB, EBCDIC</td>
</tr>
<tr>
<td>BLT8CP921</td>
<td>Latvian Standard LV88-92(1) Windows/Unix 8-bit Baltic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>LV8PC8LR</td>
<td>Latvian Version IBM-PC Code Page 866 8-bit Latin/Cyrillic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>AR8ARABICMAC</td>
<td>Mac Client 8-bit Latin/Arabic</td>
<td>SB</td>
</tr>
<tr>
<td>EE8MACCE</td>
<td>Mac Client 8-bit Central European</td>
<td>SB</td>
</tr>
<tr>
<td>EE8MACCROATIAN</td>
<td>Mac Client 8-bit Croatian</td>
<td>SB</td>
</tr>
<tr>
<td>WE8MACROMAN8</td>
<td>Mac Client 8-bit Extended Roman8 West European</td>
<td>SB</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Comments</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>EL8MACGREEK</td>
<td>Mac Client 8-bit Greek</td>
<td>SB</td>
</tr>
<tr>
<td>IS8MACICELANDIC</td>
<td>Mac Client 8-bit Icelandic</td>
<td>SB</td>
</tr>
<tr>
<td>CL8MACCYRILLIC</td>
<td>Mac Client 8-bit Latin/Cyrillic</td>
<td>SB</td>
</tr>
<tr>
<td>TR8MACTURKISH</td>
<td>Mac Client 8-bit Turkish</td>
<td>SB</td>
</tr>
<tr>
<td>TR8MACTURKISHS</td>
<td>Mac Server 8-bit Turkish</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>AR8ARABICMACS</td>
<td>Mac Server 8-bit Latin/Arabic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>EE8MACCES</td>
<td>Mac Server 8-bit Central European</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>EE8MACCROATIANS</td>
<td>Mac Server 8-bit Croatian</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>WE8MACROMAN8S</td>
<td>Mac Server 8-bit Extended Roman8 West European</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>CL8MACCYRILLICS</td>
<td>Mac Server 8-bit Latin/Cyrillic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>EL8MACGREEKS</td>
<td>Mac Server 8-bit Greek</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>IS8MACICELANDICS</td>
<td>Mac Server 8-bit Icelandic</td>
<td>SB</td>
</tr>
<tr>
<td>BG8MSWIN</td>
<td>MS Windows 8-bit Bulgarian Cyrillic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>LT8MSWIN921</td>
<td>MS Windows Code Page 921 8-bit Lithuanian</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>ET8MSWIN923</td>
<td>MS Windows Code Page 923 8-bit Estonian</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>EE8MSWIN1250</td>
<td>MS Windows Code Page 1250 8-bit East European</td>
<td>SB, ASCII, EURO</td>
</tr>
<tr>
<td>CL8MSWIN1251</td>
<td>MS Windows Code Page 1251 8-bit Latin/Cyrillic</td>
<td>SB, ASCII, EURO</td>
</tr>
<tr>
<td>WE8MSWIN1252</td>
<td>MS Windows Code Page 1252 8-bit West European</td>
<td>SB, ASCII, EURO</td>
</tr>
<tr>
<td>EL8MSWIN1253</td>
<td>MS Windows Code Page 1253 8-bit Latin/Greek</td>
<td>SB, ASCII, EURO</td>
</tr>
<tr>
<td>TR8MSWIN1254</td>
<td>MS Windows Code Page 1254 8-bit Turkish</td>
<td>SB, ASCII, EURO</td>
</tr>
<tr>
<td>BLT8MSWIN1257</td>
<td>MS Windows Code Page 1257 8-bit Baltic</td>
<td>SB, ASCII, EURO</td>
</tr>
<tr>
<td>WE8NCR4970</td>
<td>NCR 4970 8-bit West European</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>WE8NEXTSTEP</td>
<td>NeXTSTEP PostScript 8-bit West European</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>CL8KOI8R</td>
<td>RELCOM Internet Standard 8-bit Latin/Cyrillic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>US8BS2000</td>
<td>Siemens 9750-62 EBCDIC 8-bit American</td>
<td>SB</td>
</tr>
<tr>
<td>DK8BS2000</td>
<td>Siemens 9750-62 EBCDIC 8-bit Danish</td>
<td>SB</td>
</tr>
<tr>
<td>F8BS2000</td>
<td>Siemens 9750-62 EBCDIC 8-bit French</td>
<td>SB</td>
</tr>
</tbody>
</table>
### Middle Eastern Language Character Sets

Table A–6 lists the Oracle character sets that can support Middle Eastern languages. The list is alphabetical by description.
### Table A–6  Middle Eastern Character Sets

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR8APTEC715</td>
<td>APTEC 715 Server 8-bit Latin/Arabic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>AR8APTEC715T</td>
<td>APTEC 715 8-bit Latin/Arabic</td>
<td>SB</td>
</tr>
<tr>
<td>AR8ASMO708PLUS</td>
<td>ASMO 708 Plus 8-bit Latin/Arabic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>AR8ASMO8X</td>
<td>ASMO Extended 708 8-bit Latin/Arabic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>AR8ADOS710</td>
<td>Arabic MS-DOS 710 Server 8-bit Latin/Arabic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>AR8ADOS710T</td>
<td>Arabic MS-DOS 710 8-bit Latin/Arabic</td>
<td>SB</td>
</tr>
<tr>
<td>AR8ADOS720</td>
<td>Arabic MS-DOS 720 Server 8-bit Latin/Arabic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>AR8ADOS720T</td>
<td>Arabic MS-DOS 720 8-bit Latin/Arabic</td>
<td>SB</td>
</tr>
<tr>
<td>TR7DEC</td>
<td>DEC VT100 7-bit Turkish</td>
<td>SB</td>
</tr>
<tr>
<td>TR8DEC</td>
<td>DEC 8-bit Turkish</td>
<td>SB</td>
</tr>
<tr>
<td>WE8EBCDIC37C</td>
<td>EBCDIC Code Page 37 8-bit Oracle/c</td>
<td>SB</td>
</tr>
<tr>
<td>IW8EBCDIC424</td>
<td>EBCDIC Code Page 424 8-bit Latin/Hebrew</td>
<td>SB</td>
</tr>
<tr>
<td>IW8EBCDIC424S</td>
<td>EBCDIC Code Page 424 Server 8-bit Latin/Hebrew</td>
<td>SB</td>
</tr>
<tr>
<td>WE8EBCDIC500C</td>
<td>EBCDIC Code Page 500 8-bit Oracle/c</td>
<td>SB</td>
</tr>
<tr>
<td>IW8EBCDIC1086</td>
<td>EBCDIC Code Page 1086 8-bit Hebrew</td>
<td>SB</td>
</tr>
<tr>
<td>AR8EBCDIC420S</td>
<td>EBCDIC Code Page 420 Server 8-bit Latin/Arabic</td>
<td>SB</td>
</tr>
<tr>
<td>AR8EBCDICX</td>
<td>EBCDIC XBASE Server 8-bit Latin/Arabic</td>
<td>SB</td>
</tr>
<tr>
<td>TR8EBCDIC1026</td>
<td>EBCDIC Code Page 1026 8-bit Turkish</td>
<td>SB</td>
</tr>
<tr>
<td>TR8EBCDIC1026S</td>
<td>EBCDIC Code Page 1026 Server 8-bit Turkish</td>
<td>SB</td>
</tr>
<tr>
<td>AR8HPARABIC8T</td>
<td>HP 8-bit Latin/Arabic</td>
<td>SB</td>
</tr>
<tr>
<td>TR8PC857</td>
<td>IBM-PC Code Page 857 8-bit Turkish</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>IW8PC1507</td>
<td>IBM-PC Code Page 1507/862 8-bit Latin/Hebrew</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>AR8ISO8859P6</td>
<td>ISO 8859-6 Latin/Arabic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>IW8ISO8859P8</td>
<td>ISO 8859-8 Latin/Hebrew</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>AZ8ISO8859PE</td>
<td>ISO 8859-9 Latin Azerbaijani</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>WE8ISO8859P9</td>
<td>ISO 8859-9 West European &amp; Turkish</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>LA8ISO6937</td>
<td>ISO 6937 8-bit Coded Character Set for Text Communication</td>
<td>SB, ASCII</td>
</tr>
</tbody>
</table>
### Table A–6 Middle Eastern Character Sets (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>IW7IS960</td>
<td>Israeli Standard 960 7-bit Latin/Hebrew</td>
<td>SB</td>
</tr>
<tr>
<td>IW8MACHEBREW</td>
<td>Mac Client 8-bit Hebrew</td>
<td>SB</td>
</tr>
<tr>
<td>AR8ARABICMAC</td>
<td>Mac Client 8-bit Latin/Arabic</td>
<td>SB</td>
</tr>
<tr>
<td>AR8ARABICMACT</td>
<td>Mac 8-bit Latin/Arabic</td>
<td>SB</td>
</tr>
<tr>
<td>TR8MACTURKISH</td>
<td>Mac Client 8-bit Turkish</td>
<td>SB</td>
</tr>
<tr>
<td>IW8MACHEBREWS</td>
<td>Mac Server 8-bit Hebrew</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>AR8ARABICMACS</td>
<td>Mac Server 8-bit Latin/Arabic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>TR8MACTURKISHS</td>
<td>Mac Server 8-bit Turkish</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>TR8MSWIN1254</td>
<td>MS Windows Code Page 1254 8-bit Turkish</td>
<td>SB, ASCII, EURO</td>
</tr>
<tr>
<td>IW8MSWIN1255</td>
<td>MS Windows Code Page 1255 8-bit Latin/Hebrew</td>
<td>SB, ASCII, EURO</td>
</tr>
<tr>
<td>AR8MSWIN1256</td>
<td>MS Windows Code Page 1256 8-bit Latin/Arabic</td>
<td>SB, ASCII, EURO</td>
</tr>
<tr>
<td>IN8ISCII</td>
<td>Multiple-Script Indian Standard 8-bit Latin/Indian Languages</td>
<td>SB</td>
</tr>
<tr>
<td>AR8MUSSAD768</td>
<td>Mussa’d Alarabi /2 768 Server 8-bit Latin/Arabic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>AR8MUSSAD768T</td>
<td>Mussa’d Alarabi /2 768 8-bit Latin/Arabic</td>
<td>SB</td>
</tr>
<tr>
<td>AR8NAFITHA711</td>
<td>Nafitha Enhanced 711 Server 8-bit Latin/Arabic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>AR8NAFITHA711T</td>
<td>Nafitha Enhanced 711 8-bit Latin/Arabic</td>
<td>SB</td>
</tr>
<tr>
<td>AR8NAFITHA721</td>
<td>Nafitha International 721 Server 8-bit Latin/Arabic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>AR8NAFITHA721T</td>
<td>Nafitha International 721 8-bit Latin/Arabic</td>
<td>SB</td>
</tr>
<tr>
<td>AR8SAKHR706</td>
<td>SAKHR 706 Server 8-bit Latin/Arabic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>AR8SAKHR707</td>
<td>SAKHR 707 Server 8-bit Latin/Arabic</td>
<td>SB, ASCII</td>
</tr>
<tr>
<td>AR8SAKHR707T</td>
<td>SAKHR 707 8-bit Latin/Arabic</td>
<td>SB</td>
</tr>
<tr>
<td>AR8XBASIC</td>
<td>XBASIC 8-bit Latin/Arabic</td>
<td>SB</td>
</tr>
<tr>
<td>WE8BS2000L5</td>
<td>Siemens EBCDIC.DE:04.L5 8-bit West European/Turkish</td>
<td>SB</td>
</tr>
<tr>
<td>AL16UTF16</td>
<td>See &quot;Universal Character Sets&quot; on page A-19 for details</td>
<td>MB, EURO, FIXED</td>
</tr>
<tr>
<td>AL32UTF8</td>
<td>See &quot;Universal Character Sets&quot; on page A-19 for details</td>
<td>MB, ASCII, EURO</td>
</tr>
<tr>
<td>UTF8</td>
<td>See &quot;Universal Character Sets&quot; on page A-19 for details</td>
<td>MB, ASCII, EURO</td>
</tr>
<tr>
<td>UTFE</td>
<td>See &quot;Universal Character Sets&quot; on page A-19 for details</td>
<td>MB, EURO</td>
</tr>
</tbody>
</table>
Universal Character Sets

Table A–7 lists the Oracle character sets that provide universal language support. They attempt to support all languages of the world, including, but not limited to, Asian, European, and Middle Eastern languages.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL16UTF16</td>
<td>Unicode 3.2 UTF-16 Universal character set</td>
<td>MB, EURO, FIXED</td>
</tr>
<tr>
<td>AL32UTF8</td>
<td>Unicode 3.2 UTF-8 Universal character set</td>
<td>MB, ASCII, EURO</td>
</tr>
<tr>
<td>UTF8</td>
<td>Unicode 3.0 UTF-8 Universal character set, CESU-8 compliant</td>
<td>MB, ASCII, EURO</td>
</tr>
<tr>
<td>UTFE</td>
<td>EBCDIC form of Unicode 3.0 UTF-8 Universal character set (UTF-EBCDIC)</td>
<td>MB, EURO</td>
</tr>
</tbody>
</table>

Note: CESU-8 defines an encoding scheme for Unicode that is identical to UTF-8 except for its representation of supplementary characters. In CESU-8, supplementary characters are represented as six-byte sequences that result from the transformation of each UTF-16 surrogate code unit into an eight-bit form that is similar to the UTF-8 transformation, but without first converting the input surrogate pairs to a scalar value. See Unicode Technical Report #26.

See Also: Chapter 6, “Supporting Multilingual Databases with Unicode”

Character Set Conversion Support

The following character set encodings are supported for conversion only. They cannot be used as the database or national character set:

- AL16UTF16LE
- ISO2022-CN
- ISO2022-JP
- ISO2022-KR
- HZ-GB-2312

You can use these character sets as the source_char_set or dest_char_set in the CONVERT function.
See Also:

- Oracle Database SQL Reference for more information about the CONVERT function
- "The CONVERT Function" on page 9-6

Subsets and Supersets

Table A–8 lists common subset/superset relationships.

<table>
<thead>
<tr>
<th>Subset</th>
<th>Superset</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR8ADOS710</td>
<td>AR8ADOS710T</td>
</tr>
<tr>
<td>AR8ADOS720</td>
<td>AR8ADOS720T</td>
</tr>
<tr>
<td>AR8ADOS720T</td>
<td>AR8ADOS720</td>
</tr>
<tr>
<td>AR8APTEC715</td>
<td>AR8APTEC715T</td>
</tr>
<tr>
<td>AR8ARABICMACT</td>
<td>AR8ARABICMAC</td>
</tr>
<tr>
<td>AR8ISO8859P6</td>
<td>AR8ASMO708PLUS</td>
</tr>
<tr>
<td>AR8ISO8859P6</td>
<td>AR8ASMO8X</td>
</tr>
<tr>
<td>AR8MUSSAD768</td>
<td>AR8MUSSAD768T</td>
</tr>
<tr>
<td>AR8MUSSAD768T</td>
<td>AR8MUSSAD768</td>
</tr>
<tr>
<td>AR8NAFITHA711</td>
<td>AR8NAFITHA711T</td>
</tr>
<tr>
<td>AR8NAFITHA721</td>
<td>AR8NAFITHA721T</td>
</tr>
<tr>
<td>AR8SAKHR707</td>
<td>AR8SAKHR707T</td>
</tr>
<tr>
<td>AR8SAKHR707T</td>
<td>AR8SAKHR707</td>
</tr>
<tr>
<td>BLT8CP921</td>
<td>BLT8ISO8859P13</td>
</tr>
<tr>
<td>BLT8CP921</td>
<td>LT8MSWIN921</td>
</tr>
<tr>
<td>D7DEC</td>
<td>D7SIEMENS9780X</td>
</tr>
<tr>
<td>D7SIEMENS9780X</td>
<td>D7DEC</td>
</tr>
<tr>
<td>DK7SIEMENS9780X</td>
<td>N7SIEMENS9780X</td>
</tr>
<tr>
<td>I7DEC</td>
<td>I7SIEMENS9780X</td>
</tr>
<tr>
<td>I7SIEMENS9780X</td>
<td>IW8EBCDIC424</td>
</tr>
</tbody>
</table>
US7ASCII is a special case because so many other character sets are supersets of it. Table A–9 lists supersets for US7ASCII.

Table A–9  US7ASCII Supersets

<table>
<thead>
<tr>
<th>Supersets</th>
<th>Supersets</th>
<th>Supersets</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>EE8MACCES</td>
<td>NEE8ISO8859P4</td>
</tr>
<tr>
<td>AL32UTF8</td>
<td>EE8MACCROATIANS</td>
<td>RU8BESTA</td>
</tr>
<tr>
<td>AR8ADOS710</td>
<td>EE8MSWIN1250</td>
<td>RU8PC855</td>
</tr>
<tr>
<td>AR8ADOS710T</td>
<td>EE8PC852</td>
<td>RU8PC866</td>
</tr>
<tr>
<td>AR8ADOS720</td>
<td>EL8DEC</td>
<td>SE8ISO8859P3</td>
</tr>
<tr>
<td>AR8ADOS720T</td>
<td>EL8ISO8859P7</td>
<td>TH8MACTHAIS</td>
</tr>
<tr>
<td>AR8APTEC715</td>
<td>EL8MACGreeks</td>
<td>TH8TISASCII</td>
</tr>
</tbody>
</table>
### Table A–9 US7ASCII Supersets (Cont.)

<table>
<thead>
<tr>
<th>Supersets</th>
<th>Supersets</th>
<th>Supersets</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR8APTEC715T</td>
<td>EL8MSWIN1253</td>
<td>TR8DEC</td>
</tr>
<tr>
<td>AR8ARABICMACS</td>
<td>EL8PC437S</td>
<td>TR8MACTURKISHS</td>
</tr>
<tr>
<td>AR8ASMO708PLUS</td>
<td>EL8PC851</td>
<td>TR8MSWIN1254</td>
</tr>
<tr>
<td>AR8ASMO8X</td>
<td>EL8PC869</td>
<td>TR8PC857</td>
</tr>
<tr>
<td>AR8HPARABIC8T</td>
<td>ET8MSWIN923</td>
<td>US8PC437</td>
</tr>
<tr>
<td>AR8ISO8859P6</td>
<td>HU8ABMOD</td>
<td>UTF8</td>
</tr>
<tr>
<td>AR8MSWIN1256</td>
<td>HU8CWI2</td>
<td>VN8MSWIN1258</td>
</tr>
<tr>
<td>AR8MUSSAD768</td>
<td>IN8ISCI</td>
<td>VN8VN3</td>
</tr>
<tr>
<td>AR8MUSSAD768T</td>
<td>IS8PC861</td>
<td>WE8DEC</td>
</tr>
<tr>
<td>AR8NAFITHA711</td>
<td>IW8ISO8859P8</td>
<td>WE8DG</td>
</tr>
<tr>
<td>AR8NAFITHA711T</td>
<td>IW8MACHEBREWS</td>
<td>WE8ISO8859P1</td>
</tr>
<tr>
<td>AR8NAFITHA721</td>
<td>IW8MSWIN1255</td>
<td>WE8ISO8859P15</td>
</tr>
<tr>
<td>AR8NAFITHA721T</td>
<td>IW8PC1507</td>
<td>WE8ISO8859P9</td>
</tr>
<tr>
<td>AR8SAKHR706</td>
<td>JA16EUC</td>
<td>WE8MACROMAN8S</td>
</tr>
<tr>
<td>AR8SAKHR707</td>
<td>JA16JIS</td>
<td>WE8MSWIN1252</td>
</tr>
<tr>
<td>AR8SAKHR707T</td>
<td>-</td>
<td>WE8NCR4970</td>
</tr>
<tr>
<td>AZ8ISO8859PE</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BG8MSWIN</td>
<td>-</td>
<td>WE8NEXTSTEP</td>
</tr>
<tr>
<td>BG8PC437S</td>
<td>JA16VMS</td>
<td>WE8PC850</td>
</tr>
<tr>
<td>BLT8CP921</td>
<td>KO16KSC5601</td>
<td>WE8PC858</td>
</tr>
<tr>
<td>BLT8ISO8859P13</td>
<td>KO16KSCCS</td>
<td>WE8PC860</td>
</tr>
<tr>
<td>BLT8MSWIN1257</td>
<td>KO16MSWIN949</td>
<td>WE8ROMAN8</td>
</tr>
<tr>
<td>BLT8PC775</td>
<td>-</td>
<td>ZHS16CGB231280</td>
</tr>
<tr>
<td>BN8BSCII</td>
<td>LA8ISO6937</td>
<td>ZHS16GBK</td>
</tr>
<tr>
<td>CDN8PC863</td>
<td>LA8PASSPORT</td>
<td>ZHT16BIG5</td>
</tr>
<tr>
<td>CEL8ISO8859P14</td>
<td>LT8MSWIN921</td>
<td>ZHT16CCDC</td>
</tr>
<tr>
<td>CL8ISO8859P5</td>
<td>LT8PC772</td>
<td>ZHT16DBT</td>
</tr>
</tbody>
</table>
Table A–10 displays the languages and character sets that are supported by the language and character set detection in the Character Set Scanner utilities (CSSCAN and LCSSCAN) and the Globalization Development Kit (GDK).

Each language has several character sets that can be detected.

When the binary values for a language match two or more encodings that have a subset/superset relationship, the subset character set is returned. For example, if the language is German and all characters are 7-bit, then US7ASCII is returned instead of WE8MSWIN1252, WE8ISO8859P15, or WE8ISO8859P1.

When the character set is determined to be UTF-8, the Oracle character set UTF8 is returned by default unless 4-byte characters (supplementary characters) are detected within the text. If 4-byte characters are detected, then the character set is reported as AL32UTF8.

Table A–10 Languages and Character Sets Supported by CSSCAN, LCSSCAN, and GDK

<table>
<thead>
<tr>
<th>Language</th>
<th>Character Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>AL16UTF16, AL32UTF8, AR8ISO8859P6, AR8MSWIN1256, UTF8</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>AL16UTF16, AL32UTF8, CL8ISO8859P5, CL8MSWIN1251, UTF8</td>
</tr>
<tr>
<td>Catalan</td>
<td>AL16UTF16, AL32UTF8, US7ASCII, UTF8, WE8ISO8859P1, WE8ISO8859P15, WE8MSWIN1252</td>
</tr>
<tr>
<td>Croatian</td>
<td>AL16UTF16, AL32UTF8, EE8ISO8859P2, EE8MSWIN1250, UTF8</td>
</tr>
<tr>
<td>Czech</td>
<td>AL16UTF16, AL32UTF8, EE8ISO8859P2, EE8MSWIN1250, UTF8</td>
</tr>
<tr>
<td>Danish</td>
<td>AL16UTF16, AL32UTF8, US7ASCII, UTF8, WE8ISO8859P1, WE8ISO8859P15, WE8MSWIN1252</td>
</tr>
<tr>
<td>Language</td>
<td>Character Sets</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>Dutch</td>
<td>AL16UTF16, AL32UTF8, US7ASCII, UTF8, WE8ISO8859P1, WE8ISO8859P15, WE8MSWIN1252</td>
</tr>
<tr>
<td>English</td>
<td>AL16UTF16, AL32UTF8, US7ASCII, UTF8, WE8ISO8859P1, WE8ISO8859P15, WE8MSWIN1252</td>
</tr>
<tr>
<td>Estonian</td>
<td>AL16UTF16, AL32UTF8, NEE8IOS8859P4, UTF8</td>
</tr>
<tr>
<td>Finnish</td>
<td>AL16UTF16, AL32UTF8, US7ASCII, UTF8, WE8ISO8859P1, WE8ISO8859P15, WE8MSWIN1252</td>
</tr>
<tr>
<td>French</td>
<td>AL16UTF16, AL32UTF8, US7ASCII, UTF8, WE8ISO8859P1, WE8ISO8859P15, WE8MSWIN1252</td>
</tr>
<tr>
<td>German</td>
<td>AL16UTF16, AL32UTF8, US7ASCII, UTF8, WE8ISO8859P1, WE8ISO8859P15, WE8MSWIN1252</td>
</tr>
<tr>
<td>Greek</td>
<td>AL16UTF16, AL32UTF8, EL8ISO8859P7, EL8MSWIN1253, UTF8</td>
</tr>
<tr>
<td>Hebrew</td>
<td>AL16UTF16, AL32UTF8, IW8ISO8859P8, IW8MSWIN1255, UTF8</td>
</tr>
<tr>
<td>Hungarian</td>
<td>AL16UTF16, AL32UTF8, EE8ISO8859P2, EE8MSWIN1250, UTF8</td>
</tr>
<tr>
<td>Italian</td>
<td>AL16UTF16, AL32UTF8, US7ASCII, UTF8, WE8ISO8859P1, WE8ISO8859P15, WE8MSWIN1252</td>
</tr>
<tr>
<td>Japanese</td>
<td>AL16UTF16, AL32UTF8, ISO2022-JP, JA16EUC, JA16SJIS, UTF8</td>
</tr>
<tr>
<td>Korean</td>
<td>AL16UTF16, AL32UTF8, ISO2022-KR, KO16KSC5601, KO16MSWIN949, UTF8</td>
</tr>
<tr>
<td>Malay</td>
<td>AL16UTF16, AL32UTF8, US7ASCII, UTF8, WE8ISO8859P1, WE8ISO8859P15, WE8MSWIN1252</td>
</tr>
<tr>
<td>Norwegian</td>
<td>AL16UTF16, AL32UTF8, US7ASCII, UTF8, WE8ISO8859P1, WE8ISO8859P15, WE8MSWIN1252</td>
</tr>
<tr>
<td>Polish</td>
<td>AL16UTF16, AL32UTF8, EE8ISO8859P2, EE8MSWIN1250, UTF8</td>
</tr>
<tr>
<td>Portuguese</td>
<td>AL16UTF16, AL32UTF8, US7ASCII, UTF8, WE8ISO8859P1, WE8ISO8859P15, WE8MSWIN1252</td>
</tr>
<tr>
<td>Romanian</td>
<td>AL16UTF16, AL32UTF8, EE8ISO8859P2, EE8MSWIN1250, UTF8</td>
</tr>
<tr>
<td>Russian</td>
<td>AL16UTF16, AL32UTF8, CL8ISO8859P5, CL8KOI8R, CL8MSWIN1251, UTF8</td>
</tr>
<tr>
<td>Simplified Chinese</td>
<td>AL16UTF16, AL32UTF8, HZ-GB-2312, UTF8, ZHS16GBK, ZHS16CGB231280</td>
</tr>
<tr>
<td>Slovak</td>
<td>AL16UTF16, AL32UTF8, EE8ISO8859P2, EE8MSWIN1250, UTF8</td>
</tr>
<tr>
<td>Spanish</td>
<td>AL16UTF16, AL32UTF8, US7ASCII, UTF8, WE8ISO8859P1, WE8ISO8859P15, WE8MSWIN1252</td>
</tr>
</tbody>
</table>
Linguistic Sorts

Oracle offers two kinds of linguistic sorts, monolingual and multilingual. In addition, monolingual sorts can be extended to handle special cases. These special cases (represented with a prefix X) typically mean that the characters are sorted differently from their ASCII values. For example, `ch` and `ll` are treated as a single character in XSPANISH.

All of the linguistic sorts can also be performed as case-insensitive or accent-insensitive by appending `_CI` or `_AI` to the linguistic sort name.

Table A–11 lists the monolingual linguistic sorts supported by the Oracle server.

**See Also:** Table A–1, "Oracle Supported Languages" on page A-2 for a list of the default sort for each language.

### Table A–11 Monolingual Linguistic Sorts

<table>
<thead>
<tr>
<th>Basic Name</th>
<th>Extended Name</th>
<th>Special Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARABIC</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ARABIC_MATCH</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ARABIC_ABJ_SORT</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ARABIC_ABJ_MATCH</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ASCII7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>AZERBAIJANI</td>
<td>XAZERBAIJANI</td>
<td>i, I, lowercase i without dot, uppercase I with dot</td>
</tr>
<tr>
<td>BENGALI</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BIG5</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
### Table A–11  Monolingual Linguistic Sorts (Cont.)

<table>
<thead>
<tr>
<th>Basic Name</th>
<th>Extended Name</th>
<th>Special Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BULGARIAN</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CATALAN</td>
<td>XCATALAN</td>
<td>æ, AE, ß</td>
</tr>
<tr>
<td>CROATIAN</td>
<td>XCROATIAN</td>
<td>D, L, N, d, l, n, ß</td>
</tr>
<tr>
<td>CZECH</td>
<td>XCZECH</td>
<td>ch, CH, Ch, ß</td>
</tr>
<tr>
<td>CZECH_PUNCTUATION</td>
<td>XCZECH_PUNCTUATION</td>
<td>ch, CH, Ch, ß</td>
</tr>
<tr>
<td>DANISH</td>
<td>XDANISH</td>
<td>A, ß, Å, å</td>
</tr>
<tr>
<td>DUTCH</td>
<td>XDUTCH</td>
<td>ij, IJ</td>
</tr>
<tr>
<td>EBCDIC</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>EEC_EURO</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>EEC_EUROPA3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ESTONIAN</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FINNISH</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FRENCH</td>
<td>XFRENCH</td>
<td>-</td>
</tr>
<tr>
<td>GERMAN</td>
<td>XGERMAN</td>
<td>ß</td>
</tr>
<tr>
<td>GERMAN_DIN</td>
<td>XGERMAN_DIN</td>
<td>ß, ä, ö, ü, Ä, Ö, Ü</td>
</tr>
<tr>
<td>GBK</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GREEK</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>HEBREW</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>HKSCS</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>HUNGARIAN</td>
<td>XHUNGARIAN</td>
<td>cs, gy, ny, sz, ty, zs, ß, CS, Cs, GY, Gy, NY, Ny, SZ, Sz, TY, Ty, ZS, Zs</td>
</tr>
<tr>
<td>ICELANDIC</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>INDONESIAN</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ITALIAN</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LATIN</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LATVIAN</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Table A–12 lists the multilingual linguistic sorts available in Oracle. All of them include GENERIC_M (an ISO standard for sorting Latin-based characters) as a base. Multilingual linguistic sorts are used for a specific primary language together with Latin-based characters. For example, KOREAN_M sorts Korean and Latin-based characters, but it does not collate Chinese, Thai, or Japanese characters.

<table>
<thead>
<tr>
<th>Sort Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANADIAN_M</td>
<td>Canadian French sort supports reverse secondary, special expanding characters</td>
</tr>
<tr>
<td>DANISH_M</td>
<td>Danish sort supports sorting lowercase characters before uppercase characters</td>
</tr>
<tr>
<td>FRENCH_M</td>
<td>French sort supports reverse sort for secondary</td>
</tr>
</tbody>
</table>
Calendar Systems

Table A–12  Multilingual Linguistic Sorts (Cont.)

<table>
<thead>
<tr>
<th>Sort Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERIC_M</td>
<td>Generic sorting order which is based on ISO14651 and Unicode canonical equivalence rules but excluding compatible equivalence rules</td>
</tr>
<tr>
<td>JAPANESE_M</td>
<td>Japanese sort supports SJIS character set order and EUC characters which are not included in SJIS</td>
</tr>
<tr>
<td>KOREAN_M</td>
<td>Korean sort: Hangul characters are based on Unicode binary order. Hanja characters based on pronunciation order. All Hangul characters are before Hanja characters</td>
</tr>
<tr>
<td>SPANISH_M</td>
<td>Traditional Spanish sort supports special contracting characters</td>
</tr>
<tr>
<td>THAI_M</td>
<td>Thai sort supports swap characters for some vowels and consonants</td>
</tr>
<tr>
<td>SCHINESE_RADICAL_M</td>
<td>Simplified Chinese sort based on radical as primary order and number of strokes order as secondary order</td>
</tr>
<tr>
<td>SCHINESE_STROKE_M</td>
<td>Simplified Chinese sort uses number of strokes as primary order and radical as secondary order</td>
</tr>
<tr>
<td>SCHINESE_PINYIN_M</td>
<td>Simplified Chinese PinYin sorting order</td>
</tr>
<tr>
<td>TCHINESE_RADICAL_M</td>
<td>Traditional Chinese sort based on radical as primary order and number of strokes order as secondary order</td>
</tr>
<tr>
<td>TCHINESE_STROKE_M</td>
<td>Traditional Chinese sort uses number of strokes as primary order and radical as secondary order. It supports supplementary characters.</td>
</tr>
</tbody>
</table>

See Also:  Chapter 5, "Linguistic Sorting and String Searching"

Calendar Systems

By default, most territory definitions use the Gregorian calendar system. Table A–13 lists the other calendar systems supported by the Oracle server.
Figure A–1 shows how March 27, 1998 appears in Japanese Imperial.

Figure A–1: Japanese Imperial Example

```
SQL> alter session set NLS CALENDAR = 2 'Japanese Imperial';
Session altered.

SQL> alter session set NLS DATE FORMAT = 2 "’平成’YY年’MM月’DD日’"
Session altered.

SQL> select sysdate from dual;
SYSDATE
----------
平成10年03月27日
```
Table A–14 shows the time zone names from the large time zone file.

<table>
<thead>
<tr>
<th>Time Zone Name</th>
<th>Is It in the Default Time Zone File?</th>
<th>Time Zone Name</th>
<th>Is It in the Default Time Zone File?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Africa/Algiers</td>
<td>No</td>
<td>Australia/Perth</td>
<td>Yes</td>
</tr>
<tr>
<td>Africa/Cairo</td>
<td>Yes</td>
<td>Australia/Queensland</td>
<td>Yes</td>
</tr>
<tr>
<td>Africa/Casablanca</td>
<td>No</td>
<td>Australia/South</td>
<td>Yes</td>
</tr>
<tr>
<td>Africa/Ceuta</td>
<td>No</td>
<td>Australia/Sydney</td>
<td>Yes</td>
</tr>
<tr>
<td>Africa/Djibouti</td>
<td>No</td>
<td>Australia/Tasmania</td>
<td>Yes</td>
</tr>
<tr>
<td>Africa/Freetown</td>
<td>No</td>
<td>Australia/Victoria</td>
<td>Yes</td>
</tr>
<tr>
<td>Africa/Johannesburg</td>
<td>No</td>
<td>Australia/West</td>
<td>Yes</td>
</tr>
<tr>
<td>Africa/Khartoum</td>
<td>No</td>
<td>Australia/Yancowinna</td>
<td>Yes</td>
</tr>
<tr>
<td>Africa/Mogadishu</td>
<td>No</td>
<td>Brazil/Acre</td>
<td>Yes</td>
</tr>
<tr>
<td>Africa/Nairobi</td>
<td>No</td>
<td>Brazil/DeNoronha</td>
<td>Yes</td>
</tr>
<tr>
<td>Africa/Nouakchott</td>
<td>No</td>
<td>Brazil/East</td>
<td>Yes</td>
</tr>
<tr>
<td>Africa/Tripoli</td>
<td>Yes</td>
<td>Brazil/West</td>
<td>Yes</td>
</tr>
<tr>
<td>Africa/Tunis</td>
<td>No</td>
<td>CET</td>
<td>Yes</td>
</tr>
<tr>
<td>Africa/Windhoek</td>
<td>No</td>
<td>CST</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Adak</td>
<td>Yes</td>
<td>CST6CDT</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Anchorage</td>
<td>Yes</td>
<td>Canada/Atlantic</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Anguilla</td>
<td>No</td>
<td>Canada/Central</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Araguaina</td>
<td>No</td>
<td>Canada/East-Saskatchewan</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Aruba</td>
<td>No</td>
<td>Canada/Eastern</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Asuncion</td>
<td>No</td>
<td>Canada/Mountain</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Atka</td>
<td>Yes</td>
<td>Canada/Newfoundland</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Belem</td>
<td>No</td>
<td>Canada/Pacific</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Boa_Vista</td>
<td>No</td>
<td>Canada/Saskatchewan</td>
<td>Yes</td>
</tr>
<tr>
<td>Time Zone Name</td>
<td>Is It in the Default Time Zone File?</td>
<td>Time Zone Name</td>
<td>Is It in the Default Time Zone File?</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------------------</td>
<td>-------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>America/Bogota</td>
<td>No</td>
<td>Canada/Yukon</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Boise</td>
<td>No</td>
<td>Chile/Continental</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Buenos_Aires</td>
<td>No</td>
<td>Chile/EasterIsland</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Cambridge_Bay</td>
<td>No</td>
<td>Cuba</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Cancun</td>
<td>No</td>
<td>EET</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Caracas</td>
<td>No</td>
<td>EST</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Cayenne</td>
<td>No</td>
<td>EST5EDT</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Cayman</td>
<td>No</td>
<td>Egypt</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Chicago</td>
<td>Yes</td>
<td>Eire</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Chihuahua</td>
<td>No</td>
<td>Etc/GMT</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Costa Rica</td>
<td>No</td>
<td>Etc/GMT+0</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Cuiaba</td>
<td>No</td>
<td>Etc/GMT+1</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Curacao</td>
<td>No</td>
<td>Etc/GMT+10</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Dawson</td>
<td>No</td>
<td>Etc/GMT+11</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Dawson_Creek</td>
<td>No</td>
<td>Etc/GMT+12</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Denver</td>
<td>Yes</td>
<td>Etc/GMT+2</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Detroit</td>
<td>Yes</td>
<td>Etc/GMT+3</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Edmonton</td>
<td>Yes</td>
<td>Etc/GMT+4</td>
<td>Yes</td>
</tr>
<tr>
<td>America/El_Salvador</td>
<td>No</td>
<td>Etc/GMT+5</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Ensenada</td>
<td>Yes</td>
<td>Etc/GMT+6</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Fort_Wayne</td>
<td>Yes</td>
<td>Etc/GMT+7</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Fortaleza</td>
<td>No</td>
<td>Etc/GMT+8</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Godthab</td>
<td>No</td>
<td>Etc/GMT+9</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Goose_Bay</td>
<td>No</td>
<td>Etc/GMT-0</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Grand_Turk</td>
<td>No</td>
<td>Etc/GMT-1</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Guadeloupe</td>
<td>No</td>
<td>Etc/GMT-10</td>
<td>Yes</td>
</tr>
<tr>
<td>Time Zone Name</td>
<td>Is It in the Default Time Zone File?</td>
<td>Time Zone Name</td>
<td>Is It in the Default Time Zone File?</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------</td>
<td>----------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>America/Guatemala</td>
<td>No</td>
<td>Etc/GMT-11</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Guayaquil</td>
<td>No</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>America/Halifax</td>
<td>Yes</td>
<td>Etc/GMT-12</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Havana</td>
<td>Yes</td>
<td>Etc/GMT-13</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Indiana/Indianapolis</td>
<td>Yes</td>
<td>Etc/GMT-2</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Indiana/Knox</td>
<td>No</td>
<td>Etc/GMT-3</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Indiana/Marengo</td>
<td>No</td>
<td>Etc/GMT-4</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Indiana/Vevay</td>
<td>No</td>
<td>Etc/GMT-5</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Indianapolis</td>
<td>Yes</td>
<td>Etc/GMT-6</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Inuvik</td>
<td>No</td>
<td>Etc/GMT-7</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Iqaluit</td>
<td>No</td>
<td>Etc/GMT-8</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Jamaica</td>
<td>Yes</td>
<td>Etc/GMT-9</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Juneau</td>
<td>No</td>
<td>Etc/GMT0</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Knox_IN</td>
<td>No</td>
<td>Europe/Amsterdam</td>
<td>No</td>
</tr>
<tr>
<td>America/La_Paz</td>
<td>No</td>
<td>Europe/Athens</td>
<td>No</td>
</tr>
<tr>
<td>America/Lima</td>
<td>No</td>
<td>Europe/Belfast</td>
<td>No</td>
</tr>
<tr>
<td>America/Los_Angeles</td>
<td>Yes</td>
<td>Europe/Belgrade</td>
<td>No</td>
</tr>
<tr>
<td>America/Louisville</td>
<td>No</td>
<td>Europe/Berlin</td>
<td>No</td>
</tr>
<tr>
<td>America/Maceio</td>
<td>No</td>
<td>Europe/Bratislava</td>
<td>No</td>
</tr>
<tr>
<td>America/Managua</td>
<td>No</td>
<td>Europe/Brussels</td>
<td>No</td>
</tr>
<tr>
<td>America/Manaus</td>
<td>Yes</td>
<td>Europe/Bucharest</td>
<td>No</td>
</tr>
<tr>
<td>America/Martinique</td>
<td>No</td>
<td>Europe/Bucharest</td>
<td>No</td>
</tr>
<tr>
<td>America/Mazatlan</td>
<td>Yes</td>
<td>Europe/Budapest</td>
<td>No</td>
</tr>
<tr>
<td>America/Mexico_City</td>
<td>Yes</td>
<td>Europe/Copenhagen</td>
<td>No</td>
</tr>
<tr>
<td>America/Miquelon</td>
<td>No</td>
<td>Europe/Dublin</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Montevideo</td>
<td>No</td>
<td>Europe/Gibraltar</td>
<td>No</td>
</tr>
<tr>
<td>Time Zone Name</td>
<td>Is It in the Default Time Zone File?</td>
<td>Time Zone Name</td>
<td>Is It in the Default Time Zone File?</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------</td>
<td>------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>America/Montreal</td>
<td>Yes</td>
<td>Europe/Helsinki</td>
<td>No</td>
</tr>
<tr>
<td>America/Montserrat</td>
<td>No</td>
<td>Europe/Istanbul</td>
<td>Yes</td>
</tr>
<tr>
<td>America/New_York</td>
<td>Yes</td>
<td>Europe/Kaliningrad</td>
<td>No</td>
</tr>
<tr>
<td>America/Nome</td>
<td>No</td>
<td>Europe/Kiev</td>
<td>No</td>
</tr>
<tr>
<td>America/Noronha</td>
<td>Yes</td>
<td>Europe/Lisbon</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Panama</td>
<td>No</td>
<td>Europe/Ljubljana</td>
<td>No</td>
</tr>
<tr>
<td>America/Phoenix</td>
<td>Yes</td>
<td>Europe/London</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>Europe/Luxembourg</td>
<td>No</td>
</tr>
<tr>
<td>America/Porto_Velho</td>
<td>No</td>
<td>Europe/Madrid</td>
<td>No</td>
</tr>
<tr>
<td>America/Puerto_Rico</td>
<td>No</td>
<td>Europe/Minsk</td>
<td>No</td>
</tr>
<tr>
<td>America/Rankin_Inlet</td>
<td>No</td>
<td>Europe/Monaco</td>
<td>No</td>
</tr>
<tr>
<td>America/Regina</td>
<td>Yes</td>
<td>Europe/Moscow</td>
<td>Yes</td>
</tr>
<tr>
<td>America/Rio_Branco</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>America/Santiago</td>
<td>Yes</td>
<td>Europe/Oslo</td>
<td>No</td>
</tr>
<tr>
<td>America/Sao_Paulo</td>
<td>Yes</td>
<td>Europe/Paris</td>
<td>No</td>
</tr>
<tr>
<td>America/Scoresbysund</td>
<td>No</td>
<td>Europe/Prague</td>
<td>No</td>
</tr>
<tr>
<td>America/Shiprock</td>
<td>Yes</td>
<td>Europe/Riga</td>
<td>No</td>
</tr>
<tr>
<td>America/St_Johns</td>
<td>Yes</td>
<td>Europe/Rome</td>
<td>No</td>
</tr>
<tr>
<td>America/St_Thomas</td>
<td>No</td>
<td>Europe/Samara</td>
<td>No</td>
</tr>
<tr>
<td>America/Swift_Current</td>
<td>No</td>
<td>Europe/San_Marino</td>
<td>No</td>
</tr>
<tr>
<td>America/Tegucigalpa</td>
<td>No</td>
<td>Europe/Sarajevo</td>
<td>No</td>
</tr>
<tr>
<td>America/Thule</td>
<td>No</td>
<td>Europe/Simferopol</td>
<td>No</td>
</tr>
<tr>
<td>America/Thunder_Bay</td>
<td>No</td>
<td>Europe/Skopje</td>
<td>No</td>
</tr>
<tr>
<td>America/Tijuana</td>
<td>Yes</td>
<td>Europe/Sofia</td>
<td>No</td>
</tr>
<tr>
<td>America/Tortola</td>
<td>No</td>
<td>Europe/Stockholm</td>
<td>No</td>
</tr>
<tr>
<td>America/Vancouver</td>
<td>Yes</td>
<td>Europe/Tallinn</td>
<td>No</td>
</tr>
<tr>
<td>Time Zone Name</td>
<td>Is It in the Default Time Zone File?</td>
<td>Time Zone Name</td>
<td>Is It in the Default Time Zone File?</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------</td>
<td>----------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>America/Virgin</td>
<td>No</td>
<td>Europe/Tirane</td>
<td>No</td>
</tr>
<tr>
<td>America/Whitehorse</td>
<td>Yes</td>
<td>Europe/Vatican</td>
<td>No</td>
</tr>
<tr>
<td>America/Winnipeg</td>
<td>Yes</td>
<td>Europe/Vienna</td>
<td>No</td>
</tr>
<tr>
<td>America/Yellowknife</td>
<td>No</td>
<td>Europe/Vilnius</td>
<td>No</td>
</tr>
<tr>
<td>Arctic/Longyearbyen</td>
<td>No</td>
<td>Europe/Warsaw</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Aden</td>
<td>No</td>
<td>Europe/Zagreb</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Almaty</td>
<td>No</td>
<td>Europe/Zurich</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Amman</td>
<td>No</td>
<td>GB</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Anadyr</td>
<td>No</td>
<td>GB-Eire</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Aqtau</td>
<td>No</td>
<td>GMT</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Aqtobe</td>
<td>No</td>
<td>GMT+0</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Baghdad</td>
<td>No</td>
<td>GMT-0</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Bahrain</td>
<td>No</td>
<td>GMT0</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Baku</td>
<td>No</td>
<td>Greenwich</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Bangkok</td>
<td>No</td>
<td>HST</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Beirut</td>
<td>No</td>
<td>Hongkong</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Bishkek</td>
<td>No</td>
<td>Iceland</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Calcutta</td>
<td>Yes</td>
<td>Indian/Chagos</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Chungking</td>
<td>No</td>
<td>Indian/Christmas</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Dacca</td>
<td>No</td>
<td>Indian/Cocos</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Damascus</td>
<td>No</td>
<td>Indian/Mayotte</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Dubai</td>
<td>No</td>
<td>Indian/Reunion</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Gaza</td>
<td>No</td>
<td>Iran</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Harbin</td>
<td>No</td>
<td>Israel</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Hong_Kong</td>
<td>Yes</td>
<td>Jamaica</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Irkutsk</td>
<td>No</td>
<td>Japan</td>
<td>Yes</td>
</tr>
<tr>
<td>Time Zone Name</td>
<td>Is It in the Default Time Zone File?</td>
<td>Time Zone Name</td>
<td>Is It in the Default Time Zone File?</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------</td>
<td>---------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>Asia/Istanbul</td>
<td>Yes</td>
<td>Kwajalein</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Jakarta</td>
<td>No</td>
<td>Libya</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Jayapura</td>
<td>No</td>
<td>MET</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Jerusalem</td>
<td>Yes</td>
<td>MST</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Kabul</td>
<td>No</td>
<td>MST7MDT</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Kamchatka</td>
<td>No</td>
<td>Mexico/BajaNorte</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Karachi</td>
<td>No</td>
<td>Mexico/BajaSur</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Kashgar</td>
<td>No</td>
<td>Mexico/General</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Krasnoyarsk</td>
<td>No</td>
<td>NZ</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Kuala_Lumpur</td>
<td>No</td>
<td>NZ-CHAT</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Kuching</td>
<td>No</td>
<td>Navajo</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Kuwait</td>
<td>No</td>
<td>PRC</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Macao</td>
<td>No</td>
<td>PST</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Magadan</td>
<td>No</td>
<td>PST8PDT</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Manila</td>
<td>No</td>
<td>Pacific/Auckland</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Muscat</td>
<td>No</td>
<td>Pacific/Chatham</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Nicosia</td>
<td>No</td>
<td>Pacific/Easter</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Novosibirsk</td>
<td>No</td>
<td>Pacific/Fakaofo</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Omsk</td>
<td>No</td>
<td>Pacific/Fiji</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Qatar</td>
<td>No</td>
<td>Pacific/Gambier</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Rangoon</td>
<td>No</td>
<td>Pacific/Guam</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Riyadh</td>
<td>Yes</td>
<td>Pacific/Honolulu</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Saigon</td>
<td>No</td>
<td>Pacific/Johnston</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Seoul</td>
<td>Yes</td>
<td>Pacific/Kiritimati</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Shanghai</td>
<td>Yes</td>
<td>Pacific/Kwajalein</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Singapore</td>
<td>Yes</td>
<td>Pacific/Marquesas</td>
<td>No</td>
</tr>
</tbody>
</table>
### Table A–14 Time Zone Names (Cont.)

<table>
<thead>
<tr>
<th>Time Zone Name</th>
<th>Is It in the Default Time Zone File?</th>
<th>Time Zone Name</th>
<th>Is It in the Default Time Zone File?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asia/Taipei</td>
<td>Yes</td>
<td>Pacific/Midway</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Tashkent</td>
<td>No</td>
<td>Pacific/Niue</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Tbilisi</td>
<td>No</td>
<td>Pacific/Norfolk</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Tehran</td>
<td>Yes</td>
<td>Pacific/Noumea</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Tel_Aviv</td>
<td>Yes</td>
<td>Pacific/Pago_Pago</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Tokyo</td>
<td>Yes</td>
<td>Pacific/Pitcairn</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Ujung_Pandang</td>
<td>No</td>
<td>Pacific/Rarotonga</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Urumqi</td>
<td>No</td>
<td>Pacific/Saipan</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Vladivostok</td>
<td>No</td>
<td>Pacific/Samoa</td>
<td>Yes</td>
</tr>
<tr>
<td>Asia/Yakutsk</td>
<td>No</td>
<td>Pacific/Tahiti</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Yekaterinburg</td>
<td>No</td>
<td>Pacific/Tongatapu</td>
<td>No</td>
</tr>
<tr>
<td>Asia/Yerevan</td>
<td>No</td>
<td>Pacific/Wake</td>
<td>No</td>
</tr>
<tr>
<td>Atlantic/Azores</td>
<td>No</td>
<td>Poland</td>
<td>Yes</td>
</tr>
<tr>
<td>Atlantic/Bermuda</td>
<td>No</td>
<td>Portugal</td>
<td>Yes</td>
</tr>
<tr>
<td>Atlantic/Canary</td>
<td>No</td>
<td>ROC</td>
<td>Yes</td>
</tr>
<tr>
<td>Atlantic/Faeroe</td>
<td>No</td>
<td>ROK</td>
<td>Yes</td>
</tr>
<tr>
<td>Atlantic/Madeira</td>
<td>No</td>
<td>Singapore</td>
<td>Yes</td>
</tr>
<tr>
<td>Atlantic/Reykjavik</td>
<td>Yes</td>
<td>Turkey</td>
<td>Yes</td>
</tr>
<tr>
<td>Atlantic/St_Helena</td>
<td>No</td>
<td>US/Alaska</td>
<td>Yes</td>
</tr>
<tr>
<td>Atlantic/Stanley</td>
<td>No</td>
<td>US/Aleutian</td>
<td>Yes</td>
</tr>
<tr>
<td>Australia/ACT</td>
<td>Yes</td>
<td>US/Arizona</td>
<td>Yes</td>
</tr>
<tr>
<td>Australia/Adelaide</td>
<td>Yes</td>
<td>US/Central</td>
<td>Yes</td>
</tr>
<tr>
<td>Australia/Brisbane</td>
<td>Yes</td>
<td>US/East-Indiana</td>
<td>Yes</td>
</tr>
<tr>
<td>Australia/Broken_Hill</td>
<td>Yes</td>
<td>US/Eastern</td>
<td>Yes</td>
</tr>
<tr>
<td>Australia/Canberra</td>
<td>Yes</td>
<td>US/Hawaii</td>
<td>Yes</td>
</tr>
<tr>
<td>Australia/Darwin</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This section contains information about obsolete linguistic sorts, character sets, languages, and territories.

Updates to the Oracle Language and Territory Definition Files

Changes have been made to the content in some of the language and territory definition files in Oracle Database 10g. These updates are necessary to correct the legacy definitions which no longer meet the local conventions in some of the Oracle supported languages and territories. These changes include modifications to the currency symbols, month names, and group separators. One example is the local currency symbol for Brazil. This has been updated from Cr$ to R$ in Oracle Database 10g.

Please refer to the "Oracle Language and Territory definition changes" table documented in the $ORACLE_HOME/nls/data/old/data_changes.html file for a detailed list of the changes.

Oracle Database 10g customers should review their existing application code and to make sure that the correct cultural conventions that are defined in Oracle Database 10g are used.

### Table A–14 Time Zone Names (Cont.)

<table>
<thead>
<tr>
<th>Time Zone Name</th>
<th>Is It in the Default Time Zone File?</th>
<th>Time Zone Name</th>
<th>Is It in the Default Time Zone File?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia/Hobart</td>
<td>Yes</td>
<td>US/Indiana-Starke</td>
<td>No</td>
</tr>
<tr>
<td>Australia/LHI</td>
<td>Yes</td>
<td>US/Michigan</td>
<td>Yes</td>
</tr>
<tr>
<td>Australia/Lindeman</td>
<td>Yes</td>
<td>US/Mountain</td>
<td>Yes</td>
</tr>
<tr>
<td>Australia/Lord_Howe</td>
<td>Yes</td>
<td>US/Pacific</td>
<td>Yes</td>
</tr>
<tr>
<td>Australia/Melbourne</td>
<td>Yes</td>
<td>US/Pacific-New</td>
<td>Yes</td>
</tr>
<tr>
<td>Australia/North</td>
<td>Yes</td>
<td>US/Samoa</td>
<td>Yes</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>W-SU</td>
<td>Yes</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>WET</td>
<td>Yes</td>
</tr>
</tbody>
</table>

See Also: "Choosing a Time Zone File" on page 4-20
10g are being used. For customers who may not be able to make the necessary code changes to support their applications, Oracle offers Oracle9i backward compatibility by shipping a set of Oracle9i locale definition files with Oracle Database 10g. Please refer to the README.txt file in the $ORACLE_HOME/nls/data/old directory for more information.

Oracle Corporation strongly recommends that customers use the Oracle Database 10g locale definition files; Oracle9i locale definition files will be desupported in a future release.

Obsolete Linguistic Sorts

Table A–15 contains linguistic sorts that have been desupported in Oracle Database 10g.

<table>
<thead>
<tr>
<th>Obsolete Sort Name</th>
<th>Replacement Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>THAI_TELEPHONE</td>
<td>THAI_M</td>
</tr>
<tr>
<td>THAI_DICTIONARY</td>
<td>THAI_M</td>
</tr>
<tr>
<td>CANADIAN_FRENCH</td>
<td>CANADIAN_M</td>
</tr>
<tr>
<td>JAPANESE</td>
<td>JAPANESE_M</td>
</tr>
</tbody>
</table>

CIS Is No Longer the Default Territory When the Language is RUSSIAN

RUSSIA has been added as a territory in OracleDatabase 10g. It is the default territory when the language is RUSSIAN. The CIS territory, which was the default territory in previous releases when the language was RUSSIAN, is supported for backward compatibility.

YUGOSLAVIA Is a Deprecated Territory

YUGOSLAVIA is a deprecated territory in Oracle Database 10g. Use SERBIA AND MONTENEGRO instead.

New Names for Obsolete Character Sets

Before Oracle release 7.2, when a character set was renamed, the old name was usually supported along with the new name for several releases after the change. Beginning with release 7.2, the old names are no longer supported.
Table A–16 lists the affected character sets. If you reference any of these character sets in your code, then replace them with their new name.

<table>
<thead>
<tr>
<th>Old Name</th>
<th>New Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL24UTFSS</td>
<td>UTF8, AL32UTF8</td>
</tr>
<tr>
<td>AR8MSAWIN</td>
<td>AR8MSWIN1256</td>
</tr>
<tr>
<td>CL8EBCDIC875S</td>
<td>CL8EBCDIC875R</td>
</tr>
<tr>
<td>EL8EBCDIC875S</td>
<td>EL8EBCDIC875R</td>
</tr>
<tr>
<td>JVMS</td>
<td>JA16VMS</td>
</tr>
<tr>
<td>JEUC</td>
<td>JA16EUC</td>
</tr>
<tr>
<td>SJIS</td>
<td>JA16SJIS</td>
</tr>
<tr>
<td>JDBCS</td>
<td>JA16DBCS</td>
</tr>
<tr>
<td>KSC5601</td>
<td>KO16KSC5601</td>
</tr>
<tr>
<td>KDBCS</td>
<td>KO16DBCS</td>
</tr>
<tr>
<td>CGB2312-80</td>
<td>ZHS16CGB231280</td>
</tr>
<tr>
<td>CNS 11643-86</td>
<td>ZHT32EUC</td>
</tr>
<tr>
<td>JA16EUCFIXED</td>
<td>None. Replaced by new national character set. UTF8 and AL16UTF16.</td>
</tr>
<tr>
<td>ZHS32EUCFIXED</td>
<td>None. Replaced by new national character set. UTF8 and AL16UTF16.</td>
</tr>
<tr>
<td>ZHS16GBKFIXED</td>
<td>None. Replaced by new national character set. UTF8 and AL16UTF16.</td>
</tr>
<tr>
<td>JA16DBCSFIXED</td>
<td>None. Replaced by new national character set. UTF8 and AL16UTF16.</td>
</tr>
<tr>
<td>KO16DBCSFIXED</td>
<td>None. Replaced by new national character set. UTF8 and AL16UTF16.</td>
</tr>
<tr>
<td>ZHS16DBCSFIXED</td>
<td>None. Replaced by new national character set. UTF8 and AL16UTF16.</td>
</tr>
<tr>
<td>ZHS16GBK231280FIXED</td>
<td>None. Replaced by new national character set. UTF8 and AL16UTF16.</td>
</tr>
<tr>
<td>ZHT16DBCSFIXED</td>
<td>None. Replaced by new national character set. UTF8 and AL16UTF16.</td>
</tr>
<tr>
<td>KO16KSC5601FIXED</td>
<td>None. Replaced by new national character set. UTF8 and AL16UTF16.</td>
</tr>
<tr>
<td>JA16SJISFIXED</td>
<td>None. Replaced by new national character set. UTF8 and AL16UTF16.</td>
</tr>
<tr>
<td>ZHT16BIG5FIXED</td>
<td>None. Replaced by new national character set. UTF8 and AL16UTF16.</td>
</tr>
<tr>
<td>ZHT32TRISFIXED</td>
<td>None. Replaced by new national character set. UTF8 and AL16UTF16.</td>
</tr>
</tbody>
</table>
Character set CL8MSWINDOW31 has been desupported. The newer character set CL8MSWIN1251 is actually a duplicate of CL8MSWINDOW31 and includes some characters omitted from the earlier version. Change any usage of CL8MSWINDOW31 to CL8MSWIN1251 instead.

**AL24UTFFSS Character Set Desupported**

The Unicode Character Set AL24UTFFSS was desupported in Oracle9i. AL24UTFFSS was introduced in version 7 as the Unicode character set supporting UTF-8 encoding scheme based on the Unicode standard 1.1, which is now obsolete. In Oracle Database 10g, Oracle offers the Unicode database character set AL32UTF8, which is based on Unicode 3.2, and UTF8, which is based on Unicode 3.0.

The migration path for an existing AL24UTFFSS database is to upgrade to UTF8 prior to upgrading to Oracle9i. As with all migrations to a new database character set, Oracle Corporation recommends that you use the Character Set Scanner for data analysis before attempting to migrate your existing database character set to UTF8.

*See Also:* Chapter 12, "Character Set Scanner Utilities"

**Bengali Language Definition Deprecated**

Oracle’s Bengali language definition is not compatible with Unicode standards. Oracle recommends that customers use the Bangla language definition instead. Bangla was introduced in Oracle9i Database Release 1 (9.0.1).

The Bengali language definition is supported in Oracle Database 10g Release 1 (10.1), but it may be desupported in a future release.

**Czechoslovakia Territory Definition Deprecated**

Oracle recommends that customers use either Czech Republic or Slovakia territory definitions in Oracle Database 10g Release 1 (10.1). The Czechoslovakia territory definition is supported in Oracle Database 10g Release 1 (10.1), but it may be desupported in a future release.
This appendix offers an introduction to Unicode character assignments. This appendix contains:

- Unicode Code Ranges
- UTF-16 Encoding
- UTF-8 Encoding
# Unicode Code Ranges

Table B–1 contains code ranges that have been allocated in Unicode for UTF-16 character codes.

<table>
<thead>
<tr>
<th>Types of Characters</th>
<th>First 16 Bits</th>
<th>Second 16 Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>0000-007F</td>
<td>-</td>
</tr>
<tr>
<td>European, Arabic, Hebrew</td>
<td>0080-07FF</td>
<td>-</td>
</tr>
<tr>
<td>Indic, Thai, symbols, Chinese, Japanese, Korean</td>
<td>0800-0FFF</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1000 - CFFF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D000 - D7FF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F900 - FFFF</td>
<td></td>
</tr>
<tr>
<td>Private Use Area #1</td>
<td>E000 - EFFF</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>F000 - F8FF</td>
<td></td>
</tr>
<tr>
<td>Supplementary chars</td>
<td>D800 - D8BF</td>
<td>DC00 - DFFF</td>
</tr>
<tr>
<td></td>
<td>D8CO - DABF</td>
<td>DC00 - DFFF</td>
</tr>
<tr>
<td></td>
<td>DAC0 - D7BF</td>
<td>DC00 - DFFF</td>
</tr>
<tr>
<td></td>
<td>DC00 - DFFF</td>
<td></td>
</tr>
<tr>
<td>Private Use Area #2</td>
<td>DB80 - DBBF</td>
<td>DC00 - DFFF</td>
</tr>
<tr>
<td></td>
<td>DBC0 - DBFF</td>
<td>DC00 - DFFF</td>
</tr>
</tbody>
</table>

Table B–2 contains code ranges that have been allocated in Unicode for UTF-8 character codes.

<table>
<thead>
<tr>
<th>Types of Characters</th>
<th>First Byte</th>
<th>Second Byte</th>
<th>Third Byte</th>
<th>Fourth Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>00 - 7F</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>European, Arabic, Hebrew</td>
<td>C2 - DF</td>
<td>80 - BF</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Indic, Thai, symbols, Chinese, Japanese, Korean</td>
<td>E0</td>
<td>A0 - BF</td>
<td>80 - BF</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>E1 - EC</td>
<td>80 - BF</td>
<td>80 - BF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ED</td>
<td>80 - 9F</td>
<td>80 - BF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EF</td>
<td>A4 - BF</td>
<td>80 - BF</td>
<td></td>
</tr>
</tbody>
</table>
UTF-16 Encoding

As shown in Table B–1, UTF-16 character codes for some characters (Additional Chinese/Japanese/Korean characters and Private Use Area #2) are represented in two units of 16-bits. These are supplementary characters. A supplementary character consists of two 16-bit values. The first 16-bit value is encoded in the range from 0xD800 to 0xDBFF. The second 16-bit value is encoded in the range from 0xDC00 to 0xDFFF. With supplementary characters, UTF-16 character codes can represent more than one million characters. Without supplementary characters, only 65,536 characters can be represented. Oracle’s AL16UTF16 character set supports supplementary characters.

See Also: "Supplementary Characters" on page 6-3

UTF-8 Encoding

The UTF-8 character codes in Table B–2 show that the following conditions are true:

- ASCII characters use 1 byte
- European (except ASCII), Arabic, and Hebrew characters require 2 bytes
- Indic, Thai, Chinese, Japanese, and Korean characters as well as certain symbols such as the euro symbol require 3 bytes
UTF-8 Encoding

- Characters in the Private Use Area #1 require 3 bytes
- Supplementary characters require 4 bytes
- Characters in the Private Use Area #2 require 4 bytes

Oracle’s AL32UTF8 character set supports 1-byte, 2-byte, 3-byte, and 4-byte values. Oracle’s UTF8 character set supports 1-byte, 2-byte, and 3-byte values, but not 4-byte values.
**accent**
A mark that changes the sound of a character. Because the common meaning of accent is associated with the stress or prominence of the character’s sound, the preferred word in Oracle Database Globalization Support Guide is diacritic.

*See also* diacritic.

**accent-insensitive linguistic sort**
A linguistic sort that uses information only about base letters, not diacritics or case.

*See also* linguistic sort, base letter, diacritic, case.

**AL16UTF16**
The default Oracle character set for the SQL NCHAR data type, which is used for the national character set. It encodes Unicode data in the UTF-16 encoding.

*See also* national character set.

**AL32UTF8**
An Oracle character set for the SQL CHAR data type, which is used for the database character set. It encodes Unicode data in the UTF-8 encoding.

*See also* database character set.

**ASCII**
American Standard Code for Information Interchange. A common encoded 7-bit character set for English. ASCII includes the letters A-Z and a-z, as well as digits, punctuation symbols, and control characters. The Oracle character set name is US7ASCII.
**base letter**
A character without diacritics. For example, the base letter for a, A, ä, and Ä is a. See also diacritic.

**binary sorting**
Ordering character strings based on their binary coded values.

**byte semantics**
Treatment of strings as a sequence of bytes. See also character semantics and length semantics.

**canonical equivalence**
A basic equivalence between characters or sequences of characters. For example, ç is equivalent to the combination of c and ê. They cannot be distinguished when they are correctly rendered.

**case**
Refers to the condition of being uppercase or lowercase. For example, in a Latin alphabet, A is the uppercase glyph for a, the lowercase glyph.

**case conversion**
Changing a character from uppercase to lowercase or vice versa.

**case-insensitive linguistic sort**
A linguistic sort that uses information about base letters and diacritics but not case. See also base letter, case, diacritic, linguistic sort.

**character**
A character is an abstract element of text. A character is different from a glyph, which is a specific representation of a character. For example, the first character of the English upper-case alphabet can be displayed as a, A, A, and so on. These forms are different glyphs that represent the same character. A character, a character code, and a glyph are related as follows:

character --(encoding)---> character code --(font)---> glyph

For example, the first character of the English uppercase alphabet is represented in computer memory as a number. The number is called the **encoding** or the **character code**. The character code for the first character of the English uppercase alphabet is
0x41 in the ASCII encoding scheme. The character code is 0xc1 in the EBCDIC encoding scheme.

You must choose a font to display or print the character. The available fonts depend on which encoding scheme is being used. The character can be printed or displayed as A, A, or A, for example. The forms are different glyphs that represent the same character.

See also character code and glyph.

character code
A character code is a number that represents a specific character. The number depends on the encoding scheme. For example, the character code of the first character of the English uppercase alphabet is 0x41 in the ASCII encoding scheme, but it is 0xc1 in the EBCDIC encoding scheme.

See also character.

character semantics
Treatment of strings as a sequence of characters.

See also byte semantics and length semantics.

character set
A collection of elements that represent textual information for a specific language or group of languages. One language can be represented by more than one character set.

A character set does not always imply a specific character encoding scheme. A character encoding scheme is the assignment of a character code to each character in a character set.

In this manual, a character set usually does imply a specific character encoding scheme. Therefore, a character set is the same as an encoded character set in this manual.

character set migration
Changing the character set of an existing database.

character string
An ordered group of characters.

A character string can also contain no characters. In this case, the character string is called a null string. The number of characters in a null string is 0 (zero).
character classification
Character classification information provides details about the type of character associated with each character code. For example, a character can uppercase, lowercase, punctuation, or control character.

character encoding scheme
A rule that assigns numbers (character codes) to all characters in a character set. Encoding scheme, encoding method, and encoding also mean character encoding scheme.

client character set
The encoded character set used by the client. A client character set can differ from the server character set. The server character set is called the database character set. If the client character set is different from the database character set, then character set conversion must occur.
See also database character set.

code point
The numeric representation of a character in a character set. For example, the code point of A in the ASCII character set is 0x41. The code point of a character is also called the encoded value of a character.
See also Unicode code point.

code unit
The unit of encoded text for processing and interchange. The size of the code unit varies depending on the character encoding scheme. In most character encodings, a code unit is 1 byte. Important exceptions are UTF-16 and UCS-2, which use 2-byte code units, and wide character, which uses 4 bytes.
See also character encoding scheme.

collation
Ordering of character strings according to rules about sorting characters that are associated with a language in a specific locale. Also called linguistic sort.
See also linguistic sort, monolingual linguistic sort, multilingual linguistic sort, accent-insensitive linguistic sort, case-insensitive linguistic sort.
**data scanning**
The process of identifying potential problems with character set conversion and truncation of data before migrating the database character set.

**database character set**
The encoded character set that is used to store text in the database. This includes CHAR, VARCHAR2, LONG, and fixed-width CLOB column values and all SQL and PL/SQL text.

**diacritic**
A mark near or through a character or combination of characters that indicates a different sound than the sound of the character without the diacritical mark. For example, the cedilla in façade is a diacritic. It changes the sound of c.

**EBCDIC**
Extended Binary Coded Decimal Interchange Code. EBCDIC is a family of encoded character sets used mostly on IBM systems.

**encoded character set**
A character set with an associated character encoding scheme. An encoded character set specifies the number (character code) that is assigned to each character. See also character encoding scheme.

**encoded value**
The numeric representation of a character in a character set. For example, the code point of A in the ASCII character set is 0x41. The encoded value of a character is also called the code point of a character.

**font**
An ordered collection of character glyphs that provides a graphical representation of characters in a character set.

**globalization**
The process of making software suitable for different linguistic and cultural environments. Globalization should not be confused with localization, which is the process of preparing software for use in one specific locale.
glyph
A glyph (font glyph) is a specific representation of a character. A character can have many different glyphs. For example, the first character of the English uppercase alphabet can be printed or displayed as A, A, A, and so on. These forms are different glyphs that represent the same character.

See also character.

ideograph
A symbol that represents an idea. Chinese is an example of an ideographic writing system.

ISO
International Organization for Standards. A worldwide federation of national standards bodies from 130 countries. The mission of ISO is to develop and promote standards in the world to facilitate the international exchange of goods and services.

ISO 8859
A family of 8-bit encoded character sets. The most common one is ISO 8859-1 (also known as ISO Latin1), and is used for Western European languages.

ISO 14651
A multilingual linguistic sort standard that is designed for almost all languages of the world.

See also multilingual linguistic sort.

ISO/IEC 10646
A universal character set standard that defines the characters of most major scripts used in the modern world. In 1993, ISO adopted Unicode version 1.1 as ISO/IEC 10646-1:1993. ISO/IEC 10646 has two formats: UCS-2 is a 2-byte fixed-width format, and UCS-4 is a 4-byte fixed-width format. There are three levels of implementation, all relating to support for composite characters:

- Level 1 requires no composite character support.
- Level 2 requires support for specific scripts (including most of the Unicode scripts such as Arabic and Thai).
- Level 3 requires unrestricted support for composite characters in all languages.
ISO currency
The 3-letter abbreviation used to denote a local currency, based on the ISO 4217 standard. For example, USD represents the United States dollar.

ISO Latin1
The ISO 8859-1 character set standard. It is an 8-bit extension to ASCII that adds 128 characters that include the most common Latin characters used in Western Europe. The Oracle character set name is WE8ISO8859P1.

See also ISO 8859.

length semantics
Length semantics determines how you treat the length of a character string. The length can be treated as a sequence of characters or bytes.

See also character semantics and byte semantics.

linguistic index
An index built on a linguistic sort order.

linguistic sort
A ordering of strings based on requirements from a locale instead of the binary representation of the strings.

See also multilingual linguistic sort and monolingual linguistic sort.

locale
A collection of information about the linguistic and cultural preferences from a particular region. Typically, a locale consists of language, territory, character set, linguistic, and calendar information defined in NLS data files.

localization
The process of providing language-specific or culture-specific information for software systems. Translation of an application's user interface is an example of localization. Localization should not be confused with globalization, which is the making software suitable for different linguistic and cultural environments.

monolingual linguistic sort
An Oracle sort that has two levels of comparison for strings. Most European languages can be sorted with a monolingual sort, but it is inadequate for Asian languages.
See also multilingual linguistic sort.

**monolingual support**
Support for only one language.

**multibyte**
Two or more bytes.

When character codes are assigned to all characters in a specific language or a group of languages, one byte (8 bits) can represent 256 different characters. Two bytes (16 bits) can represent up to 65,536 different characters. Two bytes are not enough to represent all the characters for many languages. Some characters require 3 or 4 bytes.

One example is the UTF8 Unicode encoding. In UTF8, there are many 2-byte and 3-byte characters.

Another example is Traditional Chinese, used in Taiwan. It has more than 80,000 characters. Some character encoding schemes that are used in Taiwan use 4 bytes to encode characters.

See also single byte.

**multibyte character**
A character whose character code consists of two or more bytes under a certain character encoding scheme.

Note that the same character may have different character codes under different encoding schemes. Oracle cannot tell whether a character is a multibyte character without knowing which character encoding scheme is being used. For example, Japanese Hankaku-Katakana (half-width Katakana) characters are one byte in the JA16SJIS encoded character set, two bytes in JA16EUC, and three bytes in UTF8.

See also single-byte character.

**multibyte character string**
A character string that consists of one of the following:

- No characters (called a **null string**)
- One or more single-byte characters
- A mixture of one or more single-byte characters and one or more multibyte characters
- One or more multibyte characters
multilingual linguistic sort
An Oracle sort that uses evaluates strings on three levels. Asian languages require a multilingual linguistic sort even if data exists in only one language. Multilingual linguistic sorts are also used when data exists in several languages.

national character set
An alternate character set from the database character set that can be specified for NCHAR, NVARCHAR2, and NCLOB columns. National character sets are in Unicode only.

NLB files
Binary files used by the Locale Builder to define locale-specific data. They define all of the locale definitions that are shipped with a specific release of the Oracle database server. You can create user-defined NLB files with Oracle Locale Builder. See also Oracle Locale Builder and NLT files.

NLS
National Language Support. NLS allows users to interact with the database in their native languages. It also allows applications to run in different linguistic and cultural environments. The term is somewhat obsolete because Oracle supports global users at one time.

NLSRTL
National Language Support Runtime Library. This library is responsible for providing locale-independent algorithms for internationalization. The locale-specific information (that is, NLSDATA) is read by the NLSRTL library during run-time.

NLT files
Text files used by the Locale Builder to define locale-specific data. Because they are in text, you can view the contents.

null string
A character string that contains no characters.

Oracle Locale Builder
A GUI utility that offers a way to view, modify, or define locale-specific data. You can also create your own formats for language, territory, character set, and linguistic sort.
replacement character
A character used during character conversion when the source character is not available in the target character set. For example, ? is often used as Oracle's default replacement character.

restricted multilingual support
Multilingual support that is restricted to a group of related languages. Western European languages can be represented with ISO 8859-1, for example. If multilingual support is restricted, then Thai could not be added to the group.

SQL CHAR datatypes
Includes CHAR, VARCHAR, VARCHAR2, CLOB, and LONG datatypes.

SQL NCHAR datatypes
Includes NCHAR, NVARCHAR, NVARCHAR2, and NCLOB datatypes.

script
A collection of related graphic symbols that are used in a writing system. Some scripts can represent multiple languages, and some languages use multiple scripts. Example of scripts include Latin, Arabic, and Han.

single byte
One byte. One byte usually consists of 8 bits. When character codes are assigned to all characters for a specific language, one byte (8 bits) can represent 256 different characters.

See also multibyte.

single-byte character
A single-byte character is a character whose character code consists of one byte under a specific character encoding scheme. Note that the same character may have different character codes under different encoding schemes. Oracle cannot tell which character is a single-byte character without knowing which encoding scheme is being used. For example, the euro currency symbol is one byte in the WE8MSWIN1252 encoded character set, two bytes in AL16UTF16, and three bytes in UTF8.

See also multibyte character.
single-byte character string

A single-byte character string is a character string that consists of one of the following:

- No character (called a **null string**)
- One or more single-byte characters

supplementary characters

The first version of Unicode was a 16-bit, fixed-width encoding that used two bytes to encode each character. This allowed 65,536 characters to be represented. However, more characters need to be supported because of the large number of Asian ideograms.

Unicode 3.1 defines supplementary characters to meet this need. It uses two 16-bit code units (also known as **surrogate pairs**) to represent a single character. This allows an additional 1,048,576 characters to be defined. The Unicode 3.1 standard added the first group of 44,944 supplementary characters.

**surrogate pairs**

*See also* supplementary characters.

syllabary

Provide a mechanism for communicating phonetic information along with the ideographic characters used by languages such as Japanese.

UCS-2

A 1993 ISO/IEC standard character set. It is a fixed-width, 16-bit Unicode character set. Each character occupies 16 bits of storage. The ISO Latin1 characters are the first 256 code points, so it can be viewed as a 16-bit extension of ISO Latin1.

UCS-4

A fixed-width, 32-bit Unicode character set. Each character occupies 32 bits of storage. The UCS-2 characters are the first 65,536 code points in this standard, so it can be viewed as a 32-bit extension of UCS-2. This is also sometimes referred to as ISO-10646.

Unicode

Unicode is a universal encoded character set that allows you information from any language to be stored by using a single character set. Unicode provides a unique code value for every character, regardless of the platform, program, or language.
**Unicode database**

A database whose database character set is UTF-8.

**Unicode code point**

A value in the Unicode codespace, which ranges from 0 to 0x10FFFF. Unicode assigns a unique code point to every character.

**Unicode datatype**

A SQL NCHAR datatype (NCHAR, NVARCHAR2, and NCLOB). You can store Unicode characters in columns of these datatypes even if the database character set is not Unicode.

**unrestricted multilingual support**

The ability to use as many languages as desired. A universal character set, such as Unicode, helps to provide unrestricted multilingual support because it supports a very large character repertoire, encompassing most modern languages of the world.

**UTFE**

A Unicode 3.0 UTF-8 Oracle database character set with 6-byte supplementary character support. It is used only on EBCDIC platforms.

**UTF8**

The UTF8 Oracle character set encodes characters in one, two, or three bytes. It is for ASCII-based platforms. The UTF8 character set supports Unicode 3.0. Although specific supplementary characters were not assigned code points in Unicode until version 3.1, the code point range was allocated for supplementary characters in Unicode 3.0. Supplementary characters are treated as two separate, user-defined characters that occupy 6 bytes.

**UTF-8**

The 8-bit encoding of Unicode. It is a variable-width encoding. One Unicode character can be 1 byte, 2 bytes, 3 bytes, or 4 bytes in UTF-8 encoding. Characters from the European scripts are represented in either 1 or 2 bytes. Characters from most Asian scripts are represented in 3 bytes. Supplementary characters are represented in 4 bytes.

**UTF-16**

The 16-bit encoding of Unicode. It is an extension of UCS-2 and supports the supplementary characters defined in Unicode 3.1 by using a pair of UCS-2 code points. One Unicode character can be 2 bytes or 4 bytes in UTF-16 encoding.
Characters (including ASCII characters) from European scripts and most Asian scripts are represented in 2 bytes. Supplementary characters are represented in 4 bytes.

**wide character**
A fixed-width character format that is useful for extensive text processing because it allows data to be processed in consistent, fixed-width chunks. Wide characters are intended to support internal character processing.
Symbols

$ORACLE_HOME/nls/data directory, 1-3
$ORACLE_HOME/oracore/zoneinfo/tzdata.dat
time zone file, 4-20
$ORACLE_HOME/oracore/zoneinfo/timezone.dat
time zone file, 4-20

Numerics

7-bit encoding schemes, 2-9
8-bit encoding schemes, 2-9

A

abbreviations
   languages, A-2
abstract datatype
   creating as NCHAR, 2-20
accent, 5-14
accent-insensitive linguistic sort, 5-11
ADCS script
   migrating character sets in Real Application
      Clusters, 11-10
ADD_MONTHS SQL function, 4-16
ADO interface and Unicode, 7-41
AL16UTF16 character set, 6-6, A-19
AL24UTFSS character set, 6-6
AL32UTF8 character set, 6-6, 6-7, A-19
ALTER DATABASE CHARACTER SET
   statement, 11-10
ALTER DATABASE NATIONAL CHARACTER
   SET statement, 11-10, 11-12
ALTER SESSION statement
   SET NLS_CURRENCY clause, 3-35, 3-37
   SET NLS_LANGUAGE clause, 3-18
   SET NLS_NUMERIC_CHARACTERS
      clause, 3-33
   SET NLS_TERRITORY clause, 3-18
ALTER TABLE MODIFY statement
   migrating from CHAR to NCHAR, 11-12, 11-13
analyse_histgrm.sql script, 12-37
analyse_rule.sql script, 12-37
analyse_source.sql script, 12-37
application-locales, 8-38
Arial Unicode MS font, 13-2
array parameter
   Database Character Set Scanner, 12-12
ASCII encoding, 2-6
AT LOCAL clause, 4-25
AT TIME ZONE clause, 4-25

B

base letter, 5-7
base letters, 5-5
BFILE data
   loading into LOBs, 9-14
binary sort, 5-2
   example, 5-17
binary sorts
   case-insensitive and accent-insensitive, 5-16
binding and defining CLOB and NCLOB data in
   OCI, 7-21
binding and defining SQL CHAR datatypes in
   OCI, 7-19
binding and defining SQL NCHAR datatypes in
   OCI, 7-20
BLANK_TRIMMING parameter, 11-4
BLOBs
  creating indexes, 6-23
boundaries parameter
  Database Character Set Scanner, 12-13
byte semantics, 2-12, 3-44

C
C number format mask, 3-36
Calendar Utility, 13-17
calendars
  customizing, 13-17
  parameter, 3-27
  supported, A-28
canonical equivalence, 5-4, 5-9
capture parameter
  Database Character Set Scanner, 12-13
case, 5-2
case-insensitive linguistic sort, 5-11
CESU-8 compliance, A-19
changing the national character set, 11-12
CHAR columns
  migrating to NCHAR columns, 11-12
character data
  converting with CONVERT SQL function, 9-6
character data conversion
database character set, 11-8
character data scanning
  before character set migration, 11-7
character rearrangement, 5-10
character repertoire, 2-3
character semantics, 2-12, 3-44
character set
  changing after database creation, 2-20
  conversion, 2-17, 13-24
  customizing, 13-22
  data loss during conversion, 2-17
  detecting with Globalization Development
    Kit, 8-32
  encoding, 2-2
  national, 2-19, 6-9, 7-6
character set conversion
  between OCI client and database server, 7-15
  parameters, 3-43
character set definition
  customizing, 13-26
  guidelines for editing files, 13-25
  naming files, 13-25
character set migration
  CSALTER script, 11-8
  identifying character data conversion
    problems, 11-7
  postmigration tasks, 11-16
  scanning character data, 11-7
character sets
  AL16UTF16, 6-6
  AL24UTFFSS, 6-6
  AL32UTF8, 6-6
  Asian, A-8
  choosing, 11-2
  choosing a character set for a Unicode
database, 6-12
  choosing a national character set, 6-14
  conversion, 2-21, 9-6
  conversion using OCI, 10-7
  data loss, 11-4
  European, A-10
  ISO 8859 series, 2-7
  Middle Eastern, A-16
  migrating and the data dictionary, 12-36
  migration, 11-2
  naming, 2-11
  restrictions on character sets used to express
    names, 2-18
  supersets and subsets, A-20
  supported, A-7
  supporting different character repertoires, 2-5
  universal, A-19
  UTFE, 6-6
character type conversion
  error reporting, 3-44
characters
  available in all Oracle database character
    sets, 2-5
  context-sensitive, 5-8
  contracting, 5-8
  user-defined, 13-23
  choosing a character set, 11-2
  choosing between a Unicode database and Unicode
datatypes, 6-10
client operating system
  character set compatibility with applications, 2-16
CLOB and NCLOB data
  binding and defining in OCI, 7-21
CLOBs
  creating indexes, 6-22
code chart
  displaying and printing, 13-18
code point, 2-2
collation
  customizing, 13-32
compatibility
  client operating system and application character sets, 2-16
  composed characters, 5-8
  context-sensitive characters, 5-8
  contracting characters, 5-8
  contracting letters, 5-10
  control characters, encoding, 2-4
conversion
  between character set ID number and character set name, 9-8
CONVERT SQL function, 9-6
  character sets, A-19
convert time zones, 4-25
convertible data
  data dictionary, 12-36
converting character data
  CONVERT SQL function, 9-6
converting character data between character sets, 9-6
Coordinated Universal Time, 4-5, 4-7
creating a database with Unicode datatypes, 6-8
creating a Unicode database, 6-8
CSALTER script, 11-8, 11-10
  checking phase, 12-42
  running, 12-41
  updating phase, 12-43
CSMSCOLUMNS table, 12-39
CSMSERRORS table, 12-39
CSMS$TABLES table, 12-39
CSMIG user, 12-9
csminst.sql script
running, 12-9
CSMV$COLUMNS view, 12-44
CSMV$CONSTRAINTS view, 12-45
CSMV$ERROR view, 12-46
CSMV$INDEXES view, 12-46
CSMV$TABLES view, 12-47
currencies
  formats, 3-34
CURRENT_DATE SQL function, 4-17
CURRENT_TIMESTAMP SQL function, 4-17
customizing time zone data, 13-17

D

data conversion
  in Pro*C/C++, 7-22
  OCI driver, 7-29
  ODBC and OLE DB drivers, 7-38
  thin driver, 7-30
  Unicode Java strings, 7-29
data dictionary
  changing character sets, 12-36
  convertible or lossy data, 12-36
data dictionary views
  NLS_DATABASE_PARAMETERS, 3-11
  NLS_INSTANCE_PARAMETERS, 3-11
  NLS_SESSION_PARAMETER, 3-11
data expansion
  during data conversion, 7-17
data expansion during character set migration, 11-2
data expansion during conversion
  JDBC thin driver, 7-35
data inconsistencies causing data loss, 11-6
data loss
  caused by data inconsistencies, 11-6
  during character set migration, 11-4
  during OCI Unicode character set conversion, 7-15
  from mixed character sets, 11-7
data loss during character set conversion, 2-17
data loss during datatype conversion
  exceptions, 7-7
data truncation, 11-2
  restrictions, 11-3
database character set
  character data conversion, 11-8
  choosing, 2-14
  compatibility between client operating system
  and applications, 2-16
  performance, 2-17
Database Character Set Scanner, 12-14
  analyse_histgrm.sql script, 12-37
  analyse_rule.sql script, 12-37
  analyse_source.sql script, 12-37
  array parameter, 12-12
  boundaries parameter, 12-13
  capture parameter, 12-13
  CSM$COLUMNS table, 12-39
  CSM$ERRORS table, 12-39
  CSM$TABLES table, 12-39
  CSMV$COLUMNS view, 12-44
  CSMV$CONSTRAINTS view, 12-45
  CSMV$ERROR view, 12-46
  CSMV$INDEXES view, 12-46
  CSMV$TABLES view, 12-47
Database Scan Summary Report, 12-25
  error messages, 12-47
  exclude parameter, 12-13
  feedback parameter, 12-14
  fromnchar parameter, 12-15
  full parameter, 12-15
  help parameter, 12-15
  Individual Exception Report, 12-33
  invoking, 12-9
  lastrpt parameter, 12-16, 12-18
  maxblocks parameter, 12-18
  online help, 12-10
  performance, 12-39
  preserve parameter, 12-19
  restrictions, 12-40
  scan modes, 12-7
  suppress parameter, 12-20
  table parameter, 12-20
  tochar parameter, 12-20
  user parameter, 12-21
  userid parameter, 12-21
  views, 12-44
Database Character Set Scanner utility, 12-6
Database Scan Summary Report, 12-25

database schemas
  designing for multiple languages, 6-17
database time zone, 4-23
datatype conversion
  data loss and exceptions, 7-7
  implicit, 7-8
  SQL functions, 7-9
datatypes
  abstract, 2-19
  DATE, 4-3
  datatime, 4-2
  inserting values into datatime datatypes, 4-8
  inserting values into interval datatypes, 4-14
  interval, 4-2, 4-12
  INTERVAL DAY TO SECOND, 4-13
  INTERVAL YEAR TO MONTH, 4-13
  supported, 2-19
  TIMESTAMP, 4-5
  TIMESTAMP WITH LOCAL TIME ZONE, 4-7
  TIMESTAMP WITH TIME ZONE, 4-5
date and time parameters, 3-20
DATE datatype, 4-3
date formats, 3-20, 3-21, 9-13
  and partition bound expressions, 3-22
dates
  ISO standard, 3-28, 9-13
  NLS_DATE_LANGUAGE parameter, 3-23
datetime datatypes, 4-2
  inserting values, 4-8
datetime format parameters, 4-18
Daylight Saving Time
  Oracle support, 4-26
daylight saving time session parameter, 4-20
days
  format element, 3-24
  language of names, 3-23
DB_TZ database time zone, 4-24
DBMS_LOB PL/SQL package, 9-14
DBMS_LOB.LOADBLOBFROMFILE
  procedure, 9-15
DBMS_LOB.LOADCLOBFROMFILE
  procedure, 9-15
DBMS_REDEFINITION.CAN_REDEF_TABLE
  procedure, 11-14
DBTIMEZONE SQL function, 4-17
dest_char_set parameter, A-19

detecting language and character sets
Globalization Development Kit, 8-32
detection
supported languages and character sets, A-23
diacritic, 5-2
dynamic performance views
V$NLS_PARAMETERS, 3-12
V$NLS_VALID_VALUES, 3-12

e
encoding
control characters, 2-4
ideographic writing systems, 2-4
numbers, 2-4
phonetic writing systems, 2-4
punctuation, 2-4
symbols, 2-4
encoding schemes
7-bit, 2-9
8-bit, 2-9
fixed-width, 2-10
multibyte, 2-10
shift-sensitive variable-width, 2-10
shift-sensitive variable-width multibyte, 2-10
single-byte, 2-9
variable-width, 2-10
environment variables
ORA_SDTZ, 4-19, 4-24
ORA_TZFILE, 4-19
error messages
languages, A-4
translation, A-4
ERROR_ON_OVERLAP_TIME session parameter, 4-20
euro
Oracle support, 3-39
exclude parameter
Database Character Set Scanner, 12-13
expanding characters, 5-10
characters
expanding, 5-8
EXTRACT (datetime) SQL function, 4-17

F
feedback parameter
Database Character Set Scanner, 12-14
fixed-width multibyte encoding schemes, 2-10
fonts
Unicode, 13-2
Unicode for UNIX, 13-3
Unicode for Windows, 13-2
format elements, 9-14
C, 9-14
D, 9-14
day, 3-24
G, 9-14
IW, 9-14
IY, 9-14
L, 9-14
month, 3-24
RM, 9-13
RN, 9-14
format masks, 3-32, 9-13
formats
currency, 3-34
date, 3-21, 4-19
numeric, 3-31
time, 3-24
FROM_TZ SQL function, 4-17
fromchar parameter, 12-14
fromnchar parameter
Database Character Set Scanner, 12-15
full parameter
Database Character Set Scanner, 12-15

G
GDK
application configuration file, 8-15
GDK application configuration file, 8-36
example, 8-43
GDK application framework for J2EE, 8-12
GDK components, 8-10
GDK error messages, 8-47
GDK Java API, 8-26
GDK Java supplied packages and classes, 8-44
Index-6

GDK Localizer object, 8-19
gdkapp.xml application configuration file, 8-36
gdkapp.xml GDK application configuration file, 8-15
GENERIC_BASELETTER linguistic sort, 5-12
getString() method, 7-31
getStringWithReplacement() method, 7-32
Globalization Development Kit
Java API, 8-26
Globalization Development Kit, 8-2
application configuration file, 8-36
character set conversion, 8-29
components, 8-10
defining supported application locales, 8-20
e-mail programs, 8-35
error messages, 8-47
framework, 8-12
integrating locale sources, 8-16
Java supplied packages and classes, 8-44
locale detection, 8-17
Localizer object, 8-19
managing localized content in static files, 8-25
managing strings in JSPs and Java servlets, 8-24
non_ASCII input and output in an HTML page, 8-21
Oracle binary and linguistic sorts, 8-31
Oracle date, number, and monetary formats, 8-30
Oracle language and character set detection, 8-32
Oracle locale information, 8-27
Oracle locale mapping, 8-28
Oracle translated locale and time zone names, 8-34
supported locale resources, 8-16
globalization features, 1-5
globalization support architecture, 1-2
Greenwich Mean Time, 4-5, 4-7
guessing the language or character set, 12-2

I

IANA character sets
  mapping with ISO locales, 8-23
ideographic writing systems, encoding, 2-4
ignorable characters, 5-7
implicit datatype conversion, 7-8
indexes
  creating for documents stored as CLOBs, 6-22
  creating for multilingual document search, 6-21
  creating indexes for documents stored as BLOBs, 6-23
  partitioned, 9-12
Individual Exception Report, 12-33
initialization parameters
  NLS_DATE_FORMAT, 4-19
  NLS_TIMESTAMP_FORMAT, 4-19
  NLS_TIMESTAMP_TZ_FORMAT, 4-19
INSTR SQL function, 7-11
INSTR SQL functions, 9-6, 9-7
Internet application
  locale determination, 8-8
  monolingual, 8-2, 8-3
  multilingual, 8-3, 8-5
interval datatypes, 4-2, 4-12
  inserting values, 4-14
INTERVAL DAY TO SECOND datatype, 4-13
INTERVAL YEAR TO MONTH datatype, 4-13
ISO 8859 character sets, 2-7
ISO locales
  mapping with IANA character sets, 8-23
ISO standard
  date format, 9-13
ISO standard date format, 3-28, 9-13
ISO week number, 9-13
IW format element, 9-14
IY format element, 9-14

J

Java
  Unicode data conversion, 7-29
Java strings
    binding and defining in Unicode, 7-26
JDBC drivers
form of use argument, 7-28
JDBC OCI driver
  and Unicode, 7-4
JDBC programming
  Unicode, 7-25
JDBC Server Side internal driver
  and Unicode, 7-4
JDBC Server Side thin driver
  and Unicode, 7-4
JDBC thin driver
  and Unicode, 7-4
data expansion during conversion, 7-35
SQL CHAR data size restriction, 7-34
language
detecting with Globalization Development Kit, 8-32
language abbreviations, A-2
Language and Character Set File Scanner, 12-2
language definition
  customizing, 13-8
  overriding, 3-8
language support, 1-6
languages
  error messages, A-4
languages and character sets
  supported by LCSCSCAN, A-23
LAST_DAY SQL function, 4-16
lastrpt parameter
  Database Character Set Scanner, 12-16, 12-18
LCSCSCAN
  error messages, 12-5
LCSCSCAN, 12-2
  supported languages and character sets, 12-5, A-23
LCSCSCAN command
  BEGIN parameter, 12-3
  END parameter, 12-3
  examples, 12-4
  FILE parameter, 12-3
  HELP parameter, 12-4
  online help, 12-4
  RESULTS parameter, 12-3
  syntax, 12-2
  length semantics, 2-12, 3-44
  LENGTH SQL functions, 9-6, 9-7
LIKE conditions in SQL statements, 9-8
LIKE2 SQL condition, 9-8
LIKE4 SQL condition, 9-8
LIKEC SQL condition, 9-8
linguistic sort
  accent-insensitive, 5-11
  BINARY, 5-16
  BINARY_AI, linguistic sort
    BINARY_CI, 5-16
  case-insensitive, 5-11
  list of defaults, A-2
linguistic sort definitions
  supported, A-25
linguistic sorts
  controlling, 9-12
  customizing, 13-32
    characters with diacritics, 13-35, 13-38
    GENERIC_BASELETTER, 5-12
    levels, 5-5
    parameters, 3-41
    list parameter, 3-31
  lmsgen utility, 10-8
loading external BFILE data into LOBs, 9-14
LOBs
  loading external BFILE data, 9-14
  storing documents in multiple languages, 6-20
locale, 3-5
locale dependencies, 3-9
locale detection
  Globalization Development Kit, 8-17
locale information
  mapping between Oracle and other standards, 10-4
locale of Internet application
determining, 8-8
locale variant, 3-9
locale-charset-map, 8-37
locale-determine-rule, 8-39
LocaleMapper class, 8-35
locale-parameter-name, 8-40
LOCALTIMESTAMP SQL function, 4-17
lossy data
M
maxblocks parameter
  Database Character Set Scanner,  12-18
message-bundles,  8-41
migrating a character set
  CSALTER script,  11-8
migrating character sets in Real Application
  Clusters,  11-10
migration
  CHAR columns to NCHAR columns,  11-12
  character sets,  11-2
  to NCHAR datatypes,  11-10
  version 8 NCHAR columns to Oracle9i and later,  11-11
mixed character sets
  causing data loss,  11-7
monetary parameters,  3-33
monolingual Internet application,  8-3
monolingual linguistic sort
  example,  5-18
monolingual linguistic sorts
  supported,  A-25
months
  format element,  3-24
  language of names,  3-23
MONTHS_BETWEEN SQL function,  4-16
multibyte encoding schemes,  2-10
  fixed-width,  2-10
  shift-sensitive variable-width,  2-10
  variable-width,  2-10
multilexers
  creating,  6-21
multilingual data
  specifying column lengths,  6-18
multilingual document search
  creating indexes,  6-21
multilingual Internet application,  8-5
multilingual linguistic sort
  example,  5-18
multilingual linguistic sorts
  supported,  A-27

multilingual support
  restricted,  2-24
  unrestricted,  2-25
multiple languages
  designing database schemas,  6-17
  storing data,  6-18
  storing documents in LOBs,  6-20

N
N SQL function,  7-10
national character set,  2-19, 6-9, 7-6
  before Oracle9i,  6-9
NCHAR
  creating abstract datatype,  2-20
NCHAR columns
  migrating from version 8 to Oracle9i and later,  11-11
NCHAR datatype,  7-5
  migrating,  11-10
  migration,  11-11
NCHR SQL function,  7-12
NCLOB datatype,  7-7
NEW_TIME SQL function,  4-16
NEXT_DAY SQL function,  4-16
NLB data
  transportable,  13-42
NLB file,  13-5
NLB files,  13-2
  generating and installing,  13-40
NLS Calendar Utility,  13-17
NLS parameters
  default values in SQL functions,  9-3
  list,  3-3
  setting,  3-2
  specifying in SQL functions,  9-3
  unacceptable in SQL functions,  9-5
NLS Runtime Library,  1-2
NLS_CALENDAR parameter,  3-30
NLS_CHARSET_DECL_LEN SQL function,  9-9
NLS_CHARSET_ID SQL function,  9-9
NLS_CHARSET_NAME SQL function,  9-9
NLS_CREDIT parameter,  3-42, 9-12
NLS_CURRENCY parameter,  3-40
NLS_CURRENCY parameter,  3-34
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCIStringDisplayLength(),</td>
<td>10-6</td>
</tr>
<tr>
<td>OCIMultiByteStrncpy(),</td>
<td>10-6</td>
</tr>
<tr>
<td>OCIMultiByteStrncmp(),</td>
<td>10-6</td>
</tr>
<tr>
<td>OCIMultiByteStrncat(),</td>
<td>10-6</td>
</tr>
<tr>
<td>OCIWideCharStrcat(),</td>
<td>10-5</td>
</tr>
<tr>
<td>OCIWideCharStrchr(),</td>
<td>10-5</td>
</tr>
<tr>
<td>OCIWideCharStrcpi(),</td>
<td>10-5</td>
</tr>
<tr>
<td>OCIWideCharStrcat(),</td>
<td>10-5</td>
</tr>
<tr>
<td>OCIWideCharStrncpy(),</td>
<td>10-5</td>
</tr>
<tr>
<td>OCIWideCharStrcmp(),</td>
<td>10-5</td>
</tr>
<tr>
<td>OCIWideCharStrlen(),</td>
<td>10-6</td>
</tr>
<tr>
<td>OCIWideCharStrncat(),</td>
<td>10-5</td>
</tr>
<tr>
<td>OCIWideCharStrncpy(),</td>
<td>10-6</td>
</tr>
<tr>
<td>OCIWideCharStrchr(),</td>
<td>10-5</td>
</tr>
<tr>
<td>OCIWideCharToLower(),</td>
<td>10-5</td>
</tr>
<tr>
<td>OCIWideCharToUpper(),</td>
<td>10-5</td>
</tr>
<tr>
<td>ORA_NLS10 environment variable,</td>
<td>1-3</td>
</tr>
<tr>
<td>ORA_SDTZ environment variable,</td>
<td>4-19</td>
</tr>
<tr>
<td>ORA_TZFILE environment variable,</td>
<td>4-19</td>
</tr>
<tr>
<td>Oracle Call Interface and Unicode,</td>
<td>7-3</td>
</tr>
<tr>
<td>Oracle Data Provider for .NET and Unicode,</td>
<td>7-3</td>
</tr>
<tr>
<td>Oracle Language and Character Set Detection Java classes,</td>
<td>8-32</td>
</tr>
<tr>
<td>Oracle Locale Builder</td>
<td></td>
</tr>
<tr>
<td>Oracle ODBC driver and Unicode,</td>
<td>7-3</td>
</tr>
<tr>
<td>Oracle OLE DB driver and Unicode,</td>
<td>7-3</td>
</tr>
<tr>
<td>Oracle Pro*C/C++ and Unicode,</td>
<td>7-3</td>
</tr>
<tr>
<td>oracle.i18n.lcsd package,</td>
<td>8-45</td>
</tr>
<tr>
<td>oracle.i18n.net package,</td>
<td>8-45</td>
</tr>
<tr>
<td>oracle.i18n.Servlet package,</td>
<td>8-45</td>
</tr>
<tr>
<td>oracle.i18n.text package,</td>
<td>8-45</td>
</tr>
<tr>
<td>oracle.i18n.util package,</td>
<td>8-46</td>
</tr>
<tr>
<td>oracle.sql.CHAR class</td>
<td></td>
</tr>
<tr>
<td>character set conversion,</td>
<td>7-31</td>
</tr>
<tr>
<td>getString() method</td>
<td>7-31</td>
</tr>
<tr>
<td>getStringWithReplacement() method</td>
<td>7-32</td>
</tr>
<tr>
<td>toString() method</td>
<td>7-32</td>
</tr>
<tr>
<td>ORDER BY clause</td>
<td>9-12</td>
</tr>
<tr>
<td>OS_TZ local operating system time zone,</td>
<td>4-24</td>
</tr>
</tbody>
</table>

**P**

- page-charset, 8-38
- parameters
  - BLANK_TRIMMING, 11-4
calendar, 3-27
calendar, 3-30
character set conversion, 3-43
character set conversion, 3-44
linguistic sorts, 3-41
methods of setting, 3-3
monetary, 3-33
NLS_CALENDAR, 3-30
NLS_COMP, 3-42
NLS_CREDIT, 3-40
NLS_CURRENCY, 3-34
NLS_DATE_FORMAT, 3-21
NLS_DATE_LANGUAGE, 3-22
NLS_DEBIT, 3-40
NLS_DUAL_CURRENCY, 3-37
NLS_ISO_CURRENCY, 3-35
NLS_MONETARY_CHARACTERS, 3-31
NLS_NUMERICAL, 3-5
NLS_LANGUAGE, 3-12
NLS_LIST_SEPARATOR, 3-43
NLS_MONETARY_CHARACTERS, 3-40
NLS_NCHAR_CONV_EXCP, 3-43
NLS_NUMERIC_CHARACTERS, 3-32
NLS_SORT, 3-41
NLS_TERRITORY, 3-15
numeric, 3-31
setting, 3-2
time and date, 3-20
time zone, 3-25, 3-26
partitioned
indexes, 9-12
tables, 9-12
performance
choosing a database character set, 2-17
during OCI Unicode character set conversion, 7-16
phonetic writing systems, encoding, 2-4
PL/SQL and SQL and Unicode, 7-4
preserve parameter
Database Character Set Scanner, 12-19
primary level sort, 5-5
Private Use Area, 13-25
Pro*C/C++
data conversion, 7-22
NVARCHAR datatype, 7-24
UCHAR datatype, 7-24
VARCHAR datatype, 7-23
punctuation, encoding, 2-4

R
Real Application Clusters
database character set migration, 11-10
REGEXP SQL functions, 5-22
regular expressions
character class, 5-24
character range, 5-23
collation element delimiter, 5-23
equivalence class, 5-24
examples, 5-24
multilingual environment, 5-22
replacement characters
CONVERT SQL function, 9-6
restricted multilingual support, 2-24
restrictions
data truncation, 11-3
passwords, 11-3
space padding during export, 11-4
usernames, 11-3
reverse secondary sorting, 5-10
ROUND (date) SQL function, 4-17
RPAD SQL function, 7-11

S
scan modes
Database Character Set Scanner, 12-7
full database scan, 12-7
single table scan, 12-8
user tables scan, 12-8
scan.err file, 12-25
scan.out file, 12-22, 12-24, 12-25
scan.txt file, 12-25
searching multilingual documents, 6-21
searching string, 5-22
secondary level sort, 5-5
session parameters
ERROR_ON_OVRLAP, 4-20
session time zone, 4-24
SESSIONTIMEZONE SQL function, 4-17
setFormOfUse() method, 7-28
shift-sensitive variable-width multibyte encoding
schemes, 2-10
single-byte encoding schemes, 2-9
Index-12

sorting
- reverse secondary, 5-10

specifying nondefault linguistic sorts, 3-41, 3-43

source_char_set parameter, A-19

space padding
- during export, 11-4

special combination letters, 5-8, 5-10

special letters, 5-8, 5-10

special lowercase letters, 5-11

special uppercase letters, 5-11

SQL CHAR datatypes, 2-14

OCI, 7-19

SQL conditions
- LIKE2, 9-8
- LIKE4, 9-8
- LIKEC, 9-8

SQL functions
- ADD_MONTH, 4-16
- CONVERT, 9-6
- CURRENT_DATE, 4-17
- CURRENT_TIMESTAMP, 4-17
- datatype conversion, 7-9
- DBTIMEZONE, 4-17
- default values for NLS parameters, 9-3
- EXTRACT (datetime), 4-17
- FROM_TZ, 4-17
- INSTR, 7-11, 9-6, 9-7
- LAST_DAY, 4-16
- LENGTH, 9-6, 9-7
- LOCALTIMESTAMP, 4-17
- MONTHS_BETWEEN, 4-16
- N, 7-10
- NCHR, 7-12
- NEW_TIME, 4-16
- NEXT_DAY, 4-16
- NLS_CHARSETDECLLEN, 9-9
- NLS_CHARSETID, 9-9
- NLS_CHARSETNAME, 9-9
- NLS_INITCAP, 5-11, 9-2
- NLS_LOWER, 5-11, 9-2
- NLS_UPPER, 5-11, 5-12, 9-2
- NLSSORT, 9-2, 9-9
- NUMTODSINTERVAL, 4-17
- NUMTOYMINTERVAL, 4-17
- ROUND (date), 4-17
- RPAD, 7-11
- SESSIONTIMEZONE, 4-17
- specifying NLS parameters, 9-3
- SUBSTR, 9-6, 9-7
- SUBSTR2, 9-7
- SUBSTR4, 9-7
- SUBSTRB, 9-7
- SUBSTRC, 9-7
- SYS_EXTRACT.UTC, 4-17
- SYSDATE, 4-17
- SYSTIMESTAMP, 4-17
- TO_CHAR, 9-2
- TO_CHAR (datetime), 4-18
- TO_DATE, 7-10, 9-2
- TO_DSINTERVAL, 4-18
- TO_NCHAR, 7-10
- TO_NUMBER, 9-2
- TO_TIMESTAMP, 4-18
- TO_TIMESTAMP_TZ, 4-18
- TO_YMINTERVAL, 4-18
- TRUNC (date), 4-17
- TZ_OFFSET, 4-18
- unacceptable NLS parameters, 9-5
- UNISTR, 7-12

SQL NCHAR datatypes
- binding and defining in OCI, 7-20

SQL statements
- LIKE conditions, 9-8
- strict superset, 6-3
- string comparisons
  - WHERE clause, 9-11
- string literals
  - Unicode, 7-11

string manipulation using OCI, 10-4

strings
- searching, 5-22
- SUBSTR SQL function, 9-7
- SUBSTR SQL functions, 9-6, 9-7
- SUBSTR, 9-7
- SUBSTR2, 9-7
- SUBSTR4, 9-7
- SUBSTRB, 9-7
- SUBSTRC, 9-7
- SUBSTR4 SQL function, 9-7
- SUBSTRB SQL function, 9-7
SUBSTRC SQL function, 9-7
superset, strict, 6-3
supersets and subsets, A-20
supplementary characters, 5-4, 6-3
    linguistic sort support, A-28
supported datatypes, 2-19
supported territories, A-6
suppress parameter
    Database Character Set Scanner, 12-20
surrogate pairs, 6-3
syllabary, 2-4
symbols, encoding, 2-4
SYS_EXTRACT_UTC SQL function, 4-17
SYSDATE SQL function, 4-17
    effect of session time zone, 4-24
SYSTIMESTAMP SQL function, 4-17

T

table parameter
    Database Character Set Scanner, 12-20
tables
    partitioned, 9-12
territory
    dependencies, 3-9
territory definition, 3-15
customizing, 13-11
    overriding, 3-8
territory support, 1-7, A-6
territory variant, 3-9
tertiary level sort, 5-6
Thai and Laotian character rearrangement, 5-10
tilde, 7-36
time and date parameters, 3-20
time zone
    database, 4-23
effect on SYSDATE SQL function, 4-24
    session, 4-24
time zone abbreviations, 4-20
time zone data
    source, 4-20
time zone environment variables, 4-19
time zone file
    choosing, 4-20
default, 4-20
time zone names, 4-20
time zone parameters, 3-25, 3-26
time zones
    converting, 4-25
    customizing, 13-17
    TIMESTAMP datatype, 4-5
    when to use, 4-12
    TIMESTAMP datatypes
        choosing, 4-12
timestamp format, 3-25
    TIMESTAMP WITH LOCAL TIME ZONE
datatype, 4-7
    when to use, 4-12
    TIMESTAMP WITH TIME ZONE datatype, 4-5
    when to use, 4-12
timezlr.dat file, 13-17
timezone.dat file, 13-17
TO_CHAR (datetime) SQL function, 4-18
TO_CHAR SQL function, 9-2
    default date format, 3-21, 4-19
    format masks, 9-13
group separator, 3-32
    language for dates, 3-22
    spelling of days and months, 3-23
TO_DATE SQL function, 7-10, 9-2
    default date format, 3-21, 4-19
    format masks, 9-13
    language for dates, 3-22
    spelling of days and months, 3-23
TO_DSINTERVAL SQL function, 4-18
TO_NCHAR SQL function, 7-10
TO_NUMBER SQL function, 9-2
    format masks, 9-13
TO_TIMESTAMP SQL function, 4-18
TO_TIMESTAMP_TZ SQL function, 4-18
TO_YMINTERVAL SQL function, 4-18
tochar parameter
    Database Character Set Scanner, 12-20
toString() method, 7-32
transportable NLB data, 13-42
TRUNC (date) SQL function, 4-17
TZ_OFFSET SQL function, 4-18
TZABBREV, 4-20
TZNME, 4-20
UCS-2 encoding, 6-4
Unicode, 6-2
    binding and defining Java strings, 7-26
    character code assignments, B-2
    character set conversion between OCI client and database server, 7-15
    code ranges for UTF-16 characters, B-2
    code ranges for UTF-8 characters, B-2
    data conversion in Java, 7-29
    JDBC OCI driver, 7-4
    JDBC programming, 7-25
    JDBC Server Side internal driver, 7-4
    JDBC Server Side thin driver, 7-4
    JDBC thin driver, 7-4
    ODBC and OLE DB programming, 7-37
    Oracle Call Interface, 7-3
    Oracle Data Provide for .NET, 7-3
    Oracle ODBC driver, 7-3
    Oracle OLE DB driver, 7-3
    Oracle Pro*C/C++, 7-3
    Oracle support, 6-5
    parsing an XML stream with Java, 7-44
    PL/SQL and SQL, 7-4
    Private Use Area, 13-25
    programming, 7-2
    reading an XML file with Java, 7-43
    string literals, 7-11
    UCS-2 encoding, 6-4
    UTF-16 encoding, 6-4
    UTF-8 encoding, 6-3
    writing an XML file with Java, 7-42
    XML programming, 7-41
Unicode database, 6-7
    case study, 6-15
    choosing a character set, 6-12
    using with Unicode datatypes (case study), 6-17
    when to use, 6-10
Unicode datatypes, 6-8
    case study, 6-16
    choosing a national character set, 6-14
    using with a Unicode database (case study), 6-17
    when to use, 6-11
Unicode encoding, 6-3
Unicode fonts, 13-2
Unicode mode, 7-14
UNISTR SQL function, 7-12
unrestricted multilingual support, 2-25
url-rewrite-rule, 8-42
US7ASCII
    supersets, A-21
user parameter
    Database Character Set Scanner, 12-21
    user-defined characters, 13-23
    adding to a character set definition, 13-30
    cross-references between character sets, 13-25
userid parameter
    Database Character Set Scanner, 12-21
UTC, 4-5, 4-7
UTF-16 encoding, 6-4, B-3
UTF8 character set, 6-7, A-19
UTF-8 encoding, 6-3, B-3
UTFE character set, 6-6, 6-8, A-19
UTL_FILE package, using with NCHAR, 7-12
UTL_118N PL/SQL package, 8-47
UTL_LMS PL/SQL package, 8-47
UVARCHAR datatype
    Pro*C/C++, 7-24
V
V$NLS_PARAMETERS dynamic performance view, 3-12
V$NLS_VALID_VALUES dynamic performance view, 3-12
VARCHAR datatype
    Pro*C/C++, 7-23
    variable-width multibyte encoding schemes, 2-10
    version 8 NCHAR columns
        migrating to Oracle9i and later, 11-11
W
wave dash, 7-36
WHERE clause
    string comparisons, 9-11
XML
  parsing in Unicode with Java,  7-44
  reading in Unicode with Java,  7-43
  writing in Unicode with Java,  7-42
XML programming
  Unicode,  7-41