NLS_LANG Client Settings and JDBC Drivers [ID 115001.1]

Modified: Mar 27, 2012 Type: BULLETIN Status: PUBLISHED Priority: 3

Applies to:

Oracle Server - Enterprise Edition - Version: 8.1.5.0 and later [Release: 8.1.5 and later]
Generic UNIX

Purpose

The purpose of this article is to summarize ways to access an Oracle database from a client when a Java-based application is used, as well as to clarify NLS_LANG setting question.

Scope and Application

Oracle Support Analysts, Database Administrators.

NLS_LANG Client Settings and JDBC Drivers

NLS_LANG client setting and JDBC drivers

Users can write a Java application using JDBC or SQLJ programs with embedded SQL statements to access an Oracle database. The following are Oracle8i's Java components that provide NLS support:

1. JDBC Driver - Oracle provides JDBC as the core programmatic interface for accessing Oracle8i databases.

   There are three JDBC drivers provided by Oracle: two for client access and one for server access:
   - the JDBC OCI driver is used by Java applications
   - the JDBC Thin driver is primarily used by Java applets
   - the JDBC Server driver is a server-side driver that is used by Java classes running on the Java VM of the database server

2. SQLJ Translator - SQLJ acts like a preprocessor that translates embedded SQL in the SQLJ program file into a Java source file with JDBC calls. It gives programmers a higher level of programmatic interface for accessing databases.

3. Java Runtime Environment - A Java VM based on that of the JDK is integrated into the database server that enables the running of Java classes. It comes with a set of supporting services such as the library manager, which manages Java classes stored in the database.

4. CORBA Support - In addition to the Java runtime environment, Oracle integrates the CORBA Object Request Broker (ORB) into the database server, and makes the database a CORBA server. Any CORBA client can call the Java CORBA objects published to the ORB of the database server.

5. EJB Support - The Enterprise Java Bean version 1.0 container is built into the database server to provide a platform to develop and deploy EJBs.

This article will cover only the JDBC Driver as a means to access an Oracle database. For more information on all other components please refer to Chapter 6 of the 'Oracle8i National Language Support Guide Release 2 (8.1.6)' (Part number A76966-01).

JDBC and CHARACTERSET conversion

Oracle JDBC drivers allow users to retrieve data from or insert data into a database in any character set that Oracle supports. Please keep in in mind that the target character set on the client is always UCS2 because Java is based on UCS2 type of Unicode. Therefore character set conversion will take place to convert data from the database character set to UCS2. This applies to CHAR, LONG, CLOB, and VARCHAR2 data types; RAW and BLOB data are not converted.

Since the Java-client side is using Unicode, we strongly recommend to use an Unicode (AL32)UTF8 database.

The techniques that Oracle's drivers use to perform character set conversion for Java applications depend on the character set the database uses.

To check the database character set, run the following SQL command:

```
SQL> select value from nls_database_parameters
   2 where parameter = 'NLS_CHARACTERSET';
```

Example 1:

The database uses the US7ASCII or WE8ISO8859P1 character set. In this case, the driver converts the data directly from the database character set to UCS2, which is used in Java applications.

Example 2:
The database uses a non-US7ASCII or non-WE8ISO8859P1 character set (like JA16SJIS, JA16EUC, ZHT16BIG5...). The driver converts the data, first to UTF8, then to UCS2.

Example 3:
The database uses a AL32(UTF8) character set. The driver converts the data from UTF8 to UCS2. This is a very "cheap" operation with extremely low memory and cpu overhead.

When character data is inserted into or retrieved from the database, Oracle JDBC drivers perform character set conversions as appropriate. The drivers convert Unicode characters used by Java clients to Oracle database character set characters, and vice versa.

When used in a web (HTTP) environment make sure that your web env is correct per: NLS_LANG and webservers explained.

When using JDBC to insert data in N-types (NCHAR,NVARCHAR2,NCLOB) please:

- see point 14 in Note 227330.1 for information about oracle.jdbc.convertNcharLiterals
- check the documentation set for oracle.jdbc.defaultNChar

You have 2 kinds of JDBC: "Thick" and "Thin"

Thick JDBC

A "Thick JDBC" connection is using this syntax: "jdbc:oracle:oci8:@....."

In 9i:

When Java application uses 9i or lower Thick JDBC to communicate with the Oracle database, NLS_LANG environment variable needs to be set on the server where the Thick JDBC driver is installed.

NLS_LANG needs to be set to the characterset used in the web application or (for non-web java applications) the terminal character set.

Examples:

American_America.WE8MSWIN1252
"English_United Kingdom.UTF8"

9i Thick JDBC (=OCI) driver will make use of the NLS_LANG to determine the how to convert characters. NOT defining the NLS_LANG will make this to use the default US7ASCII setting, any non-ASCII data from /to the database will be lost.

You don't need to specify anything to your Java code for JDBC to pick up the NLS_LANG.

From 10g onwards the Thick JDBC driver is ignoring the NLS_LANG and uses Language, Territory and characterset settings from the JVM locale.

In 11g the property -Doracle.jdbc.ociNlsLangBackwardCompatible=true is settable on the command. If set it causes JDBC to get the characterset from NLS_LANG instead of getting the client characterset id from the locale.

Language and territory are taken from the locale regardless of the property though.

Thin JDBC

A JDBC Thin driver ("jdbc:oracle:thin:@.....") is not using the NLS_LANG.

The Oracle Thin driver supports the following character sets trough basic zip/jar files (classes111.zip, classes111.jar, classes12.zip and classes12.jar, ojdbc14.jar) they contain all the necessary classes to provide complete NLS support for:

- Oracle Character sets for CHAR/VARCHAR/LONGVARCHAR/CLOB type data that is not retrieved or inserted as a data member of an Oracle 8 Object or Collection type.

- NLS support for CHAR/VARCHAR data members of Objects and Collections for a few commonly used character sets. These character sets are: US7ASCII, WE8DEC, WE8ISO8859P1 and UTF8.

If your database use a other character set (WE8ISO8859P15,WE8MSWIN1252, EE8MSWIN1250,...) you must include nls_charsetxx.zip/nls_charsetxx.jar in your CLASSPATH.

If this file is not in your CLASSPATH, you will see the following exception:

```
java.sql.SQLException: Non supported character set: oracle-character-set-178
```

when connecting to a WE8MSWIN1252 database.

The "oracle-character-set-178" may be different depending on the characterset of the database you try to connect to.

Connecting to a WE8ISO8859P15 db will return "oracle-character-set-46" for example.

Please do not try to put multiple versions of the Oracle JDBC drivers in your CLASSPATH.

On Windows clients:

- Add [ORACLE_HOME]\jdbc\lib\classes111.zip and [ORACLE_HOME]\jdbc\lib\nls_charset11.zip to your CLASSPATH. (Add classes12.zip and nls_charset12.zip if JDK 1.2.x or 1.3 is used. Add ojdbc14.jar and nls_charset12.zip if JDK 1.4 is used.)
- Make sure [ORACLE_HOME]\bin is in your PATH.

See the Readme.txt in [ORACLE_HOME]\jdbc for more information.

When using an WE8MSWIN1252 database please be aware of:

Bug:4659157 NLS: ORA-942 SELECT TABLE WITH LEADING EURO SIGN CHAR IN WE8MSWIN1252 D/B
Fixed-Releases: A204 B106 WIN:A203P06
Details:Some WE8MSWIN1252 characters are not converted properly by the JDBC Thin driver.

Possible problems with "roundtrips"

Character data making 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 would be the Unicode full-width tilde character (0xFF5E) and its mapping to Oracle's JA16SJIS database 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. This issue is not a bug in Oracle's JDBC, but rather 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 by simply using a (AL32)UTF8 database.

Common java client configurations to access an Oracle database

1. Java applications running on client Java VMs - Java applications running on the Java VM of the client machine can access the database via either JDBC OCI or JDBC Thin drivers. Java applications can also be a middle tier servlet running on a Web server. The applications use JDBC drivers to invoke SQL, PL/SQL as well as Java stored procedures. The JDBC Thin and JDBC OCI drivers make sure that Java stored procedures will be running in the same locale as that of the client Java VM.

2. Java applets running in browsers - Java applets running in browsers can access the Oracle8i database via the JDBC Thin driver. No client-side Oracle library is required. The applets use the JDBC Thin driver to invoke SQL, PL/SQL as well as Java stored procedures. The JDBC Thin driver makes sure that Java stored procedures run in the same locale as that of the Java VM running the applets.
For more info about the correct configuration of a HTTP server please see Document 229786.1 NLS_LANG and webservers explained.

3. C clients such as OCI, Pro*C, and ODBC - Non-Java clients can call Java stored procedures the same way they call PL/SQL stored procedures. Client always gets messages from the server in the language specified by NLS_LANG. Data in the client are converted to and from the database character set by OCI (and is so using the NLS_LANG).

4. Java CORBA clients running an ORB - A CORBA client written in Java can access CORBA objects in the database server via IIOP. The client can be of different language environments. Upon log in, the locale of the Java VM running the CORBA client will be automatically sent to the database ORB, and is used to initialize the Java VM session running the server objects. The use of

the string data type of the server objects ensures the client and server communicate in Unicode.

JDBC and other NLS settings like NLS_DATE_FORMAT

JDBC (both thick and thin) will use the JVM locale to define NLS_LANGUAGE and NLS_TERRITORY (= NLS_SESSION_PARAMETERS).

This is done by sending "alter session set NLS_.....".

a. In 9i JDBC always sends alter session commands based on the locale. This is done once the connection has been made. First settings based on NLS_LANG are sent (thick jdbc only), then (if there is one) any log on trigger fires, after that the JDCB issues the alter session commands based on the locale.

   This means that you cannot use a after logon trigger to defined a controlled NLS_SESSION_PARAMETERS setting. (= you cannot use Document 251044.1 How to set a NLS session parameter at database or schema level for all connections?)

b. In 10G:
   - JDBC thin issues the locale based alter sessions now BEFORE a log on trigger fires , so you can use Document 251044.1
   - JDBC OCI issues alter sessions based on locale which is done AFTER any logon trigger. So you can *not* use Document 251044.1, Bug 3967004 is an enhancement request to change this behavior in the future.

Note that:

- for the 10.1.0.2 driver there where no alter sessions based on locale done. (= using always AMERICAN_AMERICA for thin or the NLS_LANG for Thick). This problem was resolved in 10.1.0.3 by unpublished bug 3534669.
- in 10.1.0.3 issues alter sessions based on locale only if the locale is other than English (which fires AFTER the trigger)

Trick

By default, the JVM uses the current locale as defined by the OS. To bypass this configuration, you specify on the command line the locale to be used:

```
java -Duser.language=en -Duser.region=US MyApplication
```

Note that there are some bugs with this:
http://bugs.sun.com/bugdatabase/view_bug.do;:YfiG?bug_id=4152725
If you have questions regarding this JAVA functionality , please contact Sun

Custom Charactersets and JDBC

Note that there are some problems with the OJVM (Oracle Java Virtual Machine) and JDBC when using a user defined characterset in 10.2.0.3 and lower.
You will see errors like:

```
ORA-29532: Java call terminated by uncaught Java exception:
java.sql.SQLException: Character Set Not Supported !!
```

If you use a user defined characterset on 10.2.0.3 then you need also to apply patch 5086162, patch 5465998 and patch 6731468 installed for a complete client/server fix.

11.1.0.6 (and higher) and 10.2.0.4 (and higher) include the fix for 5086162 and 5465998.
6731468 is only needed on 10.2.0.3
11.1.0.6 and 10.2.0.4 also include this trough 5465998

See the Globalzation manual on how generate a custom characterset for JDBC with Ginstall.

References

NOTE:158577.1 - NLS_LANG Explained (How does Client-Server Character Conversion Work?)
NOTE:229786.1 - NLS_LANG and webservers explained.
NOTE:241047.1 - The Priority of NLS Parameters Explained (Where To Define NLS Parameters)
NOTE:179133.1 - The correct NLS_LANG in a Windows Environment
NOTE:264157.1 - The correct NLS_LANG setting in Unix Environments