ITcareershift.com Blog

A Blog for the New Oracle DBA , would be Oracle DBA's interested to shift their career

itcareershift flash images will display here

Oracle Character set – Everything a New oracle DBA needs to know

Posted on February 4, 2011 by x_sridhar

A new oracle dba can find here everything about Oracle Character sets.

http://www.oracle.com/technology/tech/globalization/htdocs/nls_lang%20faq.htm#_Toc110410573

Below is the guideline from Oracle that is currently followed for the non-old(9i,10g) Databases .

Character Set AL32UTF8

National Character set AL16UTF16

1)Oracle recommends using AL32UTF8 as the character set compared to UTF8.Why?

UTF8 was the UTF-8 encoded character set in Oracle 8 and 8i. To maintain compatibility with existing installations

this character set will remain at unicode version 3.0 in future oracle releases.

Specific supplementary characters were not assigned to unicode until Unicode version 3.1. Hence the supplementary

characters(chinese,japanese,korean,gothic,old italic,mathematical,musical) could come with a inverted question mark(?).

Hence oracle recommends using the AL32UTF8 for full supplementary character support. AL32UTF8 supports the latest

version of the Unicode standard.

NLS_LANG FAQ

As an FAQ you can easily navigate to the sections of interest. It is recommended to start with the NLS_LANG Parameter Fundamentals section first to get a basic understanding of how the NLS_LANG parameter works.

NLS_LANG Parameter Fundamentals

Common NLS_LANG Myths

Checking the current NLS_LANG Setting

The Priority of NLS Parameters related to NLS_LANG

Session Parameters

Instance Parameters

Database Parameters

An example of a wrong NLS_LANG setup

How to setup the NLS_LANG Properly for UNIX

How to setup the NLS_LANG Properly for Windows and DOS Code Pages

Where to set the NLS_LANG in Windows

Determine your Windows ANSI code page

The correct NLS_LANG for Windows Command Line Operations

List of common NLS_LANG settings used in the Windows Registry:

List of common NLS_LANG settings used in the Command Prompt (DOS box)

Other Frequently asked questions regarding NLS_LANG

What does the LANGUAGE component of the NLS_LANG parameter control?

What does the TERRITORY component of the NLS_LANG parameter control?

How to see what's really stored in the database?

Where is the Character Conversion Done?

Windows SQL*Plus is not showing all my extended characters?

I get a question mark or inverted question mark when selecting back just inserted characters?

Is iSQL*Plus the only unicode enabled client we support?

What about command line tools like SQL*Loader, Import, Export, utilities?

What about database links?

What about Multiple Homes on Windows?

Is there an Oracle Unicode Client on Windows?

What is a Character set or Code Page?

Why Are There Different Character sets?

What is the difference between 7 bit, 8 bit and Unicode Character sets?

How to choose the right database character set?

NLS_LANG Parameter Fundamentals

A locale is a set of information addressing linguistic and cultural requirements that corresponds to a given language and country. Traditionally, the data associated with a locale provides support for formatting and parsing of dates, times, numbers, and currencies, etc. Providing current and correct locale data has historically been the responsibility of each platform owner or vendor, leading to inconsistencies and errors in locale data.

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 indicates the client's character set, which corresponds to the character set for data to be 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*

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.

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 remainder of this document will focus on the charset component of the NLS_LANG setting, as it is the least understood and most important piece to set correctly.

Top of the Document

Common NLS_LANG Myths

- Setting the NLS_LANG to the character set of the database MAY be correct but IS often not correct. DO NOT assume that NLS_LANG needs to be the same as the database character set. THIS IS OFTEN NOT TRUE.
  · The character set defined with the NLS_LANG parameter does NOT CHANGE your client's character set. It is used to let Oracle know what character set you are USING on the client side, so Oracle can do the proper conversion. You cannot change the character set of your client by using a different NLS_LANG!

  · If you don't set the NLS_LANG on the client it uses the NLS_LANG of the server. This is also NOT true! For example, if the Oracle Installer does not populate NLS_LANG, and it is not otherwise set then its value by default is AMERICAN_AMERICA.US7ASCII. The language is AMERICAN, the territory is AMERICA, and the character set is US7ASCII.

·	Setting the LANGUAGE and TERRITORY parameters of NLS_LANG has nothing to do with the ability to store characters in a database. A NLS_LANG set to JAPANESE_JAPAN.WE8MSWIN1252 will not allow you to store Japanese, as WE8MSWIN1252 doesn't support Japanese characters. However a NLS_LANG set to AMERICAN_AMERICA.JA16SJIS will allow you to store Japanese providing the input data is truly JA16SJIS and if the database is also in a character set that can store Japanese like UTF8 or JA16SJIS)

Top of the Document

Checking the current NLS_LANG Setting

In many cases the NLS_LANG has been already set during the Oracle install or thereafter manually. To be sure you can use these methods to get back the value of NLS_LANG for SQL*Plus:

On UNIX:

SQL> HOST ECHO $NLS_LANG

This returns the value of the parameter.

On Windows:

On Windows you have two possible options, normally the NLS_LANG is set in the registry, but it can also be set in the environment, however this is not often done. The value in the environment takes precedence over the value in the registry and is used for ALL Oracle_Homes on the server. Also note that any USER environment variable takes precedence over any SYSTEM environment variable (this is Windows behavior, and has nothing to do with Oracle) if set.

To check if it's set in the environment:

```
SQL> HOST ECHO %NLS_LANG%
```

If this reports just %NLS_LANG% back, the variable is not set in the environment.

If it's set it reports something like

```
ENGLISH_UNITED KINGDOM.WE8ISO8859P1
```

If NLS_LANG is not set in the environment, check the value in the registry:

SQL>@.[%NLS_LANG%].

If you get something like:

```
Unable to open file.[ENGLISH_UNITED KINGDOM.WE8ISO8859P1].
```

The "file name" between the braces is the value of the registry parameter.

If you get this as result:

Unable to open file ".[%NLS_LANG%]." then the parameter NLS_LANG is also not set in the registry.

Note the @.[%NLS_LANG%]. technique reports the NLS_LANG known by the SQL*Plus executable, it will not read the registry itself. But if you run the HOST command first and the NLS_LANG is not set in the environment then you can be sure the variable is set in the registry if the @.[%NLS_LANG%]. returns a valid value.

All other NLS parameters can be retrieved by a:

```
SELECT * FROM NLS_SESSION_PARAMETERS;
```

Note:

SELECT USERENV (‘language’) FROM DUAL; gives the session’s <Language>_<territory> but the DATABASE character set not the client, so the value returned is not the client’s complete NLS_LANG setting!

Top of the Document

The Priority of NLS Parameters related to NLS_LANG

This section explains the order in which NLS parameters are taken into account in the database client/server model. (This does NOT cover Thin JDBC connections)

There are 3 levels at which you can set NLS parameters: Database, Instance and

Session. If a parameter is defined at more than one level then the rules on which one takes precedence are quite straightforward:

1. NLS database settings are superseded by NLS instance settings

2. NLS database & NLS instance settings are superseded by NLS session settings

Session Parameters

SELECT * from NLS_SESSION_PARAMETERS;

These are the settings used for the current SQL session.

These reflect (in this order):

1) The values of NLS parameters set by "ALTER SESSION "

ALTER SESSION set NLS_DATE_FORMAT = 'DD/MM/YYYY';

2) If there is no explicit "ALTER SESSION " statement done then it reflects the setting of the corresponding NLS parameter on the client derived from the NLS_LANG variable.

3) If NLS_LANG is specified with only the <Territory> part then AMERICAN is used as default <Language>.

So if you set NLS_LANG=_BELGIUM. WE8MSWIN1252 then you get this:

PARAMETER VALUE

—————————— —————

NLS_LANGUAGE AMERICAN

NLS_TERRITORY BELGIUM

NLS_CURRENCY <euro sign here>

NLS_ISO_CURRENCY BELGIUM

….

Note:

The difference between NLS_LANG=_BELGIUM.WE8MSWIN1252 (correct) and

NLS_LANG=BELGIUM.WE8MSWIN1252 (incorrect), you need to set the "_" as separator.

4) If NLS_LANG is specified with only the <Language> part then the <Territory> defaults to a setting based on <Language>.

So if you set NLS_LANG=ITALIAN_.WE8MSWIN1252 then you get this:

PARAMETER VALUE

——————————— —————

NLS_LANGUAGE ITALIAN

NLS_TERRITORY ITALY

NLS_CURRENCY <euro sign here>

NLS_ISO_CURRENCY ITALY

…..

Note:

Note the difference between NLS_LANG=ITALIAN_.WE8MSWIN1252 (correct) and

NLS_LANG=ITALIAN.WE8MSWIN1252 (incorrect), you need to set the "_" as separator.

5) If NLS_LANG is specified without the <Language>_<Territory> part then the <Language>_<Territory> part defaults to AMERICAN_AMERICA.

So if you set NLS_LANG=.WE8MSWIN1252 then you get this:

PARAMETER VALUE

——————————— ———-

NLS_LANGUAGE AMERICAN

NLS_TERRITORY AMERICA

NLS_CURRENCY $

NLS_ISO_CURRENCY AMERICA

….

Note:

The difference between NLS_LANG=.WE8MSWIN1252 (correct) and

NLS_LANG=WE8MSWIN1252 (incorrect), you need to set the "." as separator.

6) If the NLS_LANG is set (either like in point 3, 4 or 5) then parameters like

NLS_SORT, NLS_DATE_FORMAT, etc. can be set as a "standalone" setting and will overrule the defaults derived from NLS_LANG <Language>_<Territory> part.

So if you set NLS_LANG=AMERICAN_AMERICA.WE8MSWIN1252 and NLS_ISO_CURRENCY=FRANCE then you get this:

PARAMETER VALUE

——————————— ————

NLS_LANGUAGE AMERICAN

NLS_TERRITORY AMERICA

NLS_CURRENCY $

NLS_ISO_CURRENCY FRANCE

…

Defaults:

———

* If NLS_DATE_LANGUAGE or NLS_SORT are not set then they are derived from

NLS_LANGUAGE.

* If NLS_CURRENCY, NLS_DUAL_CURRENCY, NLS_ISO_CURRENCY, NLS_DATE_FORMAT, NLS_TIMESTAMP_FORMAT, NLS_TIMESTAMP_TZ_FORMAT, NLS_NUMERIC_CHARACTERS are not set then they are derived from NLS_TERRITORY

7) If the NLS_LANG is not set at all, then it defaults to

<Language>_<Territory>.US7ASCII and the values for the

<Language>_<Territory> part used are the ones found in

NLS_INSTANCE_PARAMETERS. Parameters like NLS_SORT defined as "standalone" on the client side are ignored.

Note:

* If set, client parameters (NLS_SESSION_PARAMETERS) always take precedence over NLS_INSTANCE_PARAMETERS and NLS_DATABASE_PARAMETERS.

* This behavior cannot be disabled on/from the server, so a parameter set on the client

always has precedence above an instance or database parameter.

* NLS_LANG cannot be changed by ALTER SESSION, NLS_LANGUAGE and NLS_TERRITORY can. However NLS_LANGUAGE and /or NLS_TERRITORY cannot be set as "standalone" parameters in the environment or registry on the client.

* NLS_SESSION_PARAMETERS is NOT visible for other sessions. If you need to trace this then you have to use a logon trigger to create your own logging table (based on session parameters)

* The <clients characterset> part of NLS_LANG is NOT shown in any system table or view.

* On Windows you have two possible options, normally the NLS_LANG is set in the registry, but it can also be set in the environment, however this is not often done and generally not recommended to do so. The value in the environment takes precedence over the value in the registry and is used for ALL Oracle_Homes on the server if defined as a system environment variable.

* NLS_LANGUAGE in the session parameters also declares the language for the client error messages.

* You cannot "set" a NLS parameter in an SQL script; you need to use ALTER SESSION.

Top of the Document

Instance Parameters

SELECT * from NLS_INSTANCE_PARAMETERS;

These are the settings in the init.ora of the database at the moment that the database was started or set through ALTER SYSTEM.

If the parameter is not explicitly set in the init.ora or defined by ALTER SYSTEM then its value is NOT derived from a "higher" parameter (we are talking about parameters like NLS_SORT that derive a default from NLS_LANGUAGE in NLS_SESSION_PARAMETERS, this is NOT the case for NLS_INSTANCE_PARAMETERS)

Note:

* NLS_LANG is not an init.ora parameter; NLS_LANGUAGE and NLS_TERRITORY are so you need to set NLS_LANGUAGE and NLS_TERRITORY separately.

* You cannot define the <clients characterset> or NLS_LANG in the init.ora

The client characterset is defined by the NLS_LANG on the client OS (see above).

* You cannot define the database characterset in the init.ora. The database characterset is defined by the "Create Database" command.

* These settings take precedence above the NLS_DATABASE_PARAMETERS.

* These values are used for the NLS_SESSION_PARAMETERS if the client the

NLS_LANG is NOT set.

* Oracle strongly recommends that you set the NLS_LANG on the client at least to

NLS_LANG=.<clients characterset>

* The NLS_LANGUAGE in the instance parameters also declares the language for the server error messages in alert.log and in trace files.

Top of the Document

Database Parameters

SELECT * from NLS_DATABASE_PARAMETERS;

Defaults to AMERICAN_AMERICA if there are no parameters explicitly set in the init.ora during database creation time. If there is parameters set in the init.ora during database creation you see them here. There is no way to change these after the database creation. Do NOT attempt to update system tables to bypass these settings! These settings are used to give the database a default if the INSTANCE and SESSION parameters are not set.

Note:

* NLS_LANG is not an init.ora parameter, NLS_LANGUAGE and NLS_TERRITORY are.

So you need to set NLS_LANGUAGE and NLS_TERRITORY separately.

* These parameters are overridden by NLS_INSTANCE_PARAMETERS and NLS_SESSION_PARAMETERS.

* You cannot define the <clients character set> or NLS_LANG in the init.ora. The client character set is defined by the NLS_LANG on the client OS.

* You cannot define the database character set in the init.ora.

The database (national) character set NLS_(NCHAR)_CHARACTERSET) is defined by the "Create Database" command.

* The NLS_CHARACTERSET and NLS_NCHAR_CHARACTERSET parameters cannot be overridden by instance or session parameters.

They are defined by the value specified in the "CREATE DATABASE command and are not intended to be changed afterwards dynamically. Do NOT update system tables to change the character set. This can corrupt your database and potentially make it impossible to open the database again.

* Setting the NLS_LANG during the creation of the database does not influence the NLS_DATABASE_PARAMETERS.

* The NLS_LANG set during the database creation has NO impact on the database National Characterset.

Additional SELECT statements:

A) SELECT name,value$ from sys.props$ where name like '%NLS%';

This gives the same info as NLS_DATABASE_PARAMETERS.

You should use NLS_DATABASE_PARAMETERS instead of props$.

Note the UPPERCASE '%NLS%'

B) SELECT * from v$nls_parameters;

This view shows the current session parameters and the *DATABASE* characterset as seen in the NLS_DATABASE_PARAMETERS view.

C) SELECT name,value from v$parameter where name like '%NLS%';

This view gives the same information as NLS_INSTANCE_PARAMETERS.

Note the LOWERCASE '%NLS%'

D) SELECT userenv ('language') from dual;

and

SELECT sys_context('userenv','language') from dual;

Both these SELECT statements give the session's <Language>_<territory> and the

DATABASE character set. The database character set is not the same as the character set of the NLS_LANG that you started this connection with! So don't be fooled, although the output of this query looks like the value of a NLS_LANG variable, it is NOT.

E) SELECT userenv ('lang') from dual;

This SELECT gives the short code that Oracle uses for the Language defined by NLS_LANGUAGE setting for this session. If NLS_LANGUAGE is set to French then this will return "F", if NLS_LANGUAGE is set to English then this will return "GB"

If NLS_LANGUAGE is set to American then this will return "US", and so on...

F) SHOW parameter NLS%

This will give the same as the NLS_INSTANCE_PARAMETERS

Top of the Document

An example of a wrong NLS_LANG setup

A database is created on a UNIX system with the US7ASCII character set. A Windows client connecting to the database works with the WE8MSWIN1252 character set (regional settings -> Western Europe /ACP 1252) and the DBA, use the UNIX shell (ROMAN8) to work on the database. The NLS_LANG is set to american_america.US7ASCII on the clients and the server.

Note:

This is an INCORRECT setup to explain character set conversion, don't use it in your environment!

A very important point (as mentioned before):

When the client NLS_LANG character set is set to the same value as the database character set, Oracle assumes that the data being sent or received are of the same (correct) encoding, so no conversions or validations may occur for performance reasons. The data is just stored as delivered by the client, bit by bit.

From Windows insert an 'é' (LATIN SMALL LETTER E WITH ACUTE) into a table NLS_TEST containing one column 'TEST' of the type 'char'.

As long as you insert into and select from the column on Windows with the WE8MSWIN1252 character set everything runs smoothly. No conversion is done and 8 bits are inserted and read back, even if the character set of the database is defined as 7 bits. This happens because a byte is 8 bits and Oracle is ALWAYS using 8 bits even with a 7 bit character set. In a correct setup the most Significant Bit is just not used and only 7 bits are taken into account.

For one reason or another you need to insert from the UNIX server. When you SELECT from tables where data is inserted by the Windows clients you get a ‘Ò’ (LATIN CAPITAL LETTER O WITH TILDE) instead of the ‘é’.

If you insert ‘é’ on the UNIX server and you SELECT the row at the Windows client you get an ‘Å’ (LATIN CAPITAL LETTER A WITH RING ABOVE) back.

Bottom line is that you have INCORRECT data in the database. You store the numeric value for ‘é’ of the WE8MSWIN1252 character set in the database but you tell Oracle this is US7ASCII data, so Oracle is NOT converting anything and just stores the numeric value (again: Oracle thinks that the client is giving US7ASCII codes because the NLS_LANG is set to US7ASCII, and the database character set is also US7ASCII -> no conversion done).

When you SELECT the same column back on the UNIX server, Oracle is again expecting that the value is correct and passes the value to the UNIX terminal without any conversion.

Now the problem is that in the WE8MSWIN1252 character set the ‘é’ has the hexadecimal value ‘E9’and in the Roman8 character set the hexadecimal value for ‘é’ is ‘C5′. Oracle just passes the value stored in the database (‘E9′) to the UNIX terminal, and the UNIX terminal thinks this is the letter ‘?’ because in its (Roman8) character set the hexadecimal value ‘E9′ is representing the letter ‘Ò’. So instead of the ‘é’ you get ‘Ò’ on the UNIX terminal screen.

The inverse (the insert on the UNIX and the SELECT on the Windows client) is the same story, but you get other results.

The solution is creating the database with a character set that contains ‘é’

(WE8MSWIN1252, WE8ISO89859P1, UTF-8, etc.) and setting the NLS_LANG on the client to WE8MSWIN1252 and on the server to WE8ROMAN8. If you then insert an ‘é’ on both sides, you will get an ‘é’ back regardless of where you SELECT them. Oracle knows then that a hexadecimal value of ‘C5’ inserted by the UNIX and an ‘E9’ from a WE8MSWIN1252 client are both ‘é’ and inserts ‘é’ into the database (the code in the database depends on the character set you have chosen).

You don’t have to switch between UNIX, Windows or other OS clients to run into this kind of problem. The same problem appears if you add Windows clients that are using another character set and have an incorrect NLS_LANG set.

Top of the Document

How to setup the NLS_LANG Properly for UNIX

To specify the locale behavior of your client Oracle software, you have to set your NLS_LANG parameter. It sets the language, territory and also the character set of your client. You need to check the locale environment settings to set your NLS_LANG 3rd field (character set) in accordance with it. To do this, use the "locale" command like this:

$ locale

Example of output:

```
LANG=fr_FR
LC_CTYPE="fr_FR.iso885915@euro"
LC_COLLATE="fr_FR.iso885915@euro"
LC_MONETARY="fr_FR.iso885915@euro"
LC_NUMERIC="fr_FR.iso885915@euro"
LC_TIME="fr_FR.iso885915@euro"
LC_MESSAGES="fr_FR.iso885915@euro"
LC_ALL=fr_FR.iso885915@euro
```

The output of this command is not exactly the same on all the Unix environments. On some platforms, it can be useful to use the following syntax to have more details about the codepage really used:

$ locale LC_CTYPE | head

```
Example of output in a HP-UX environment:
```

```
""
```

```
""
```

```
"iso885915"
```

```
""
```

Example of output in a Linux environment:

```
upper;lower;alpha;digit;xdigit;space;print;graph;blank;cntrl;
```

```
punct;alnum;
```

```
combining;combining_level3
toupper;tolower;totitle
16
1
ISO-8859-15
70
84
1
0
1
```

In these cases, the NLS_LANG 3rd field should be set to WE8ISO8859P15. On Solaris, AIX, TRU64, this syntax doesn’t give interesting complementary information. To find more details about these settings:
On Solaris, look in /usr/lib/locale
On AIX, look in /usr/lib/nls/README
On TRU64, look in /usr/lib/nls
On HP-UX, look in /usr/lib/nls/config
On Linux, look in /usr/share/locale/locale.alias

To set a chosen value for these "locale" settings, it’s needed to know which values are available. To know that, use the following syntax:

```
$ locale -a
```

Then, when you have chosen a value, for example UTF-8 on Linux, you can set it like this:

```
$ export LC_ALL=UTF-8
```

or

```
% setenv LC_ALL UTF-8
```

Example of output after the setenv:

```
$ locale

LANG=fr_FR
LC_CTYPE="UTF-8"
LC_NUMERIC="UTF-8"
LC_TIME="UTF-8"
```

```
LC_COLLATE="UTF-8"
LC_MONETARY="UTF-8"
LC_MESSAGES="UTF-8"
LC_PAPER="UTF-8"
LC_NAME="UTF-8"
LC_ADDRESS="UTF-8"
LC_TELEPHONE="UTF-8"
LC_MEASUREMENT="UTF-8"
LC_IDENTIFICATION="UTF-8"
LC_ALL=UTF-8

$ locale LC_CTYPE | head
upper;lower;alpha;digit;xdigit;space;print;

graph;blank;cntrl;punct;alnum;combining;combining_level3
toupper;tolower;totitle
16
6
UTF-8
70
84
1
0
1
```

In this case, the 3rd field (character set) of NLS_LANG should be set to UTF8.

% setenv NLS_LANG American_America.UTF8

Top of the Document

How to setup the NLS_LANG Properly for Windows and DOS Code Pages

On Windows systems, the encoding scheme (character set) is specified by a code page. Code pages are defined to support specific languages or groups of languages, which share common writing systems. From Oracle point of view the terms code page and character set mean the same. Note that in non Chinese-Japanese-Korean environments, the Windows GUI and DOS command prompt do not use the same code page.

As a result Windows uses 2 different character sets for the ANSI (sqlplusw.exe) and the OEM (dos box – sqlplus.exe) environments.

Where to set the NLS_LANG in Windows

In the Registry:

On Windows systems, you should make sure that you have set an NLS_LANG registry subkey for each of your Oracle Homes:

You can easily modify this subkey with the Windows Registry Editor:

Start -> Run...

Type "regedit", and click "ok"

Edit the following registry entry:

For Oracle version 7:

HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE

For Oracle Database versions 8, 8i and 9i:

HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\HOMEx\

where "x" is the unique number identifying the Oracle home.

HOME0 is the first installation

For Oracle Database 10g:

HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\KEY_<oracle_home_name>

There you have an entry with name NLS_LANG

When starting an Oracle tools, like SQL*Plusw, it will read the content of the oracle.key file located in the same directory to determine which registry tree will be used, therefore which NLS_LANG subkey will be used.

Note:

Some people are confused by finding a NLS_LANG set to "NA" in HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE when no version 7 was installed. This is used for backwards compatibility, and can be ignored.

As a System or User Environment Variable, in System properties:

Although the Registry is the primary repository for settings on Windows, it is not the only place where parameters can be set. Even if not at all recommended, you can set the NLS_LANG as a System or User Environment Variable in the System properties.

This setting will be used for ALL Oracle homes.

To check and modify them:

Right-click the ‘My Computericon -> ‘Properties’

Select the ‘Advanced Tab -> Click on ‘Environment Variables’

The ‘User Variables list contains the settings for the specific OS user currently logged on and the ‘System variables system-wide variables for all users.

Since these environment variables take precedence over the parameters already set in your Registry, you should not set Oracle parameters at this location unless you have a very good reason.

As an Environment variable defined in the command prompt:

Before using an Oracle command line tool you need to MANUALLY SET the NLS_LANG parameter. In an MS-DOS command prompt, use the set command, for example:

C:\> set NLS_LANG=american_america.WE8PC850

Top of the Document

Determine your Windows ANSI code page

Now that you know what the NLS_LANG is currently set to you can check to see if it properly agrees with the current ANSI code page. The ACP (ANSI Code Page) is defined by the "default locale" setting of Windows, so if you have a UK Windows 2000 client and you want to input Cyrillic (Russian) you need to change the ACP (by changing the "default locale") in order to be able to input Russian.

You’ll find its value in the registry:

Start -> Run...

Type "regedit", and click "ok"

Browse the following registry entry:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\NLS\CodePage\

There you have (all the way down) an entry with as name ACP. The value of ACP is your current GUI Codepage, for the mapping to the Oracle name. Since there are many registry entries with very similar names, please make sure that you are looking at the right place in the registry.

Additionally, the following URL provides a list of the default code pages for all Windows versions:

http://www.microsoft.com/globaldev/reference/ (under the REFERENCE tab on the left of the page)

OEM = the command line codepage, ANSI = the GUI codepage

Note that the Honk Kong HKSCS is listed here: http://www.microsoft.com/hk/hkscs/

Find the correspondent Oracle client character set:

Find the Oracle client character set in the table below based on the ACP you found above. Note that there is only ONE CORRECT value for a given ACP.

ANSI CodePage (ACP)	**Oracle Client character set (3rd part of NLS_LANG)**
1250	EE8MSWIN1250
1251	CL8MSWIN1251
1252	WE8MSWIN1252
1253	EL8MSWIN1253
1254	TR8MSWIN1254
1255	IW8MSWIN1255
1256	AR8MSWIN1256
1257	BLT8MSWIN1257
1258	VN8MSWIN1258
874	TH8TISASCII
932	JA16SJIS
936	ZHS16GBK
949	KO16MSWIN949
950	ZHT16MSWIN950 – except for Hong Kong (see below)

This is the character set used by the GUI SQL*Plus (sqlplusW.exe/ plus80W.exe / plus33W.exe) that you start through the Windows start menu. Please note the difference between the GUI SQL*Plus and the "DOS mode" SQL*Plus.

You can use UTF8 as Oracle client character set (=NLS_LANG) on Windows NT, 2000 and XP but you will be limited to use only client programs that explicitly support this configuration. This is because the user interface of Win32 is not UTF8, therefore the client program have to perform explicit conversions between UTF8 (used on Oracle side) and UTF16 (used on Win32 side).

Set it in your Registry:

Use the Windows Registry Editor to set up the NLS_LANG in your Oracle Home with the value you have just found above.

Top of the Document

The correct NLS_LANG for Windows Command Line Operations

MS-DOS mode uses, with a few exceptions like CJK, (Japanese, Korean, Simplified Chinese, and Traditional Chinese) a different code page (called OEM code page) than Windows GUI (ANSI code page). Meaning that before using an Oracle command line tool such as SQL*Plus (sqlplus.exe/ plus80.exe / plus33.exe) en svrmgrl in a command prompt then you need to MANUALLY SET the NLS_LANG parameter as an environment variable with the set DOS command BEFORE using the tool.

For Japanese, Korean, Simplified Chinese, and Traditional Chinese, the MS-DOS OEM code page (CJK) is identical to the ANSI code page meaning that, in this particular case, there is no need to set the NLS_LANG parameter in MS-DOS mode.

In all other cases, you need to set it in order to overwrite the NLS_LANG registry key already matching the ANSI code page. The new "MS-DOS dedicated" NLS_LANG needs to match the MS-DOS OEM code page that could be retrieved by typing chcp in a Command Prompt:

```
C:\> chcp
```

```
Active code page: 437
```

```
C:\> set NLS_LANG=american_america.US8PC437
```

If the NLS_LANG parameter for the MS-DOS mode session is not set appropriately, error messages and data can be corrupted due to incorrect character set conversion.

Use the following list to find the Oracle character set that fits to your MS-DOS code page in use on your locale system:

MS-DOS codepage	Oracle Client character set (3rd part of NLS_LANG)
437	US8PC437
737	EL8PC737
850	WE8PC850
852	EE8PC852
857	TR8PC857

858	WE8PC858
861	IS8PC861
862	IW8PC1507
865	N8PC865
866	RU8PC866

Top of the Document

List of common NLS_LANG settings used in the Windows Registry:

Note: this is the correct setting for the GUI SQL*Plus version, (sqlplusW.exe/ plus80W.exe / plus33W.exe)

if you are testing with "special" characters please DO use the GUI and not the "DOS box" sqlplus.exe !

Operating System Locale	**NLS_LANG Value**
Arabic (U.A.E.)	ARABIC_UNITED ARAB EMIRATES.AR8MSWIN1256
Bulgarian	BULGARIAN_BULGARIA.CL8MSWIN1251
Catalan	CATALAN_CATALONIA.WE8MSWIN1252
Chinese (PRC)	SIMPLIFIED CHINESE_CHINA.ZHS16GBK
Chinese (Taiwan)	TRADITIONAL CHINESE_TAIWAN.ZHT16MSWIN950
Chinese (Hong Kong HKCS)	TRADITIONAL CHINESE_HONG KONG.ZHT16HKSCS
Chinese (Hong Kong HKCS2001)	TRADITIONAL CHINESE_HONG KONG.ZHT16HKSCS2001 (new in 10gR1)
Croatian	CROATIAN_CROATIA.EE8MSWIN1250
Czech	CZECH_CZECH REPUBLIC.EE8MSWIN1250
Danish	DANISH_DENMARK.WE8MSWIN1252
Dutch (Netherlands)	DUTCH_THE NETHERLANDS.WE8MSWIN1252
Dutch (Belgium)	DUTCH_BELGIUM.WE8MSWIN1252
English (United Kingdom)	ENGLISH_UNITED KINGDOM.WE8MSWIN1252
English (United	AMERICAN_AMERICA.WE8MSWIN1252

States)	
Estonian	ESTONIAN_ESTONIA.BLT8MSWIN1257
Finnish	FINNISH_FINLAND.WE8MSWIN1252
French (Canada)	CANADIAN FRENCH_CANADA.WE8MSWIN1252
French (France)	FRENCH_FRANCE.WE8MSWIN1252
German (Germany)	GERMAN_GERMANY.WE8MSWIN1252
Greek	GREEK_GREECE.EL8MSWIN1253
Hebrew	HEBREW_ISRAEL.IW8MSWIN1255
Hungarian	HUNGARIAN_HUNGARY.EE8MSWIN1250
Icelandic	ICELANDIC_ICELAND.WE8MSWIN1252
Indonesian	INDONESIAN_INDONESIA.WE8MSWIN1252
Italian (Italy)	ITALIAN_ITALY.WE8MSWIN1252
Japanese	JAPANESE_JAPAN.JA16SJIS
Korean	KOREAN_KOREA.KO16MSWIN949
Latvian	LATVIAN_LATVIA.BLT8MSWIN1257
Lithuanian	LITHUANIAN_LITHUANIA.BLT8MSWIN1257
Norwegian	NORWEGIAN_NORWAY.WE8MSWIN1252
Polish	POLISH_POLAND.EE8MSWIN1250
Portuguese (Brazil)	BRAZILIAN PORTUGUESE_BRAZIL.WE8MSWIN1252
Portuguese (Portugal)	PORTUGUESE_PORTUGAL.WE8MSWIN1252
Romanian	ROMANIAN_ROMANIA.EE8MSWIN1250
Russian	RUSSIAN_CIS.CL8MSWIN1251
Slovak	SLOVAK_SLOVAKIA.EE8MSWIN1250
Spanish (Spain)	SPANISH_SPAIN.WE8MSWIN1252
Swedish	SWEDISH_SWEDEN.WE8MSWIN1252
Thai	THAI_THAILAND.TH8TISASCII
Spanish (Mexico)	MEXICAN SPANISH_MEXICO.WE8MSWIN1252
Spanish (Venezuela)	LATIN AMERICAN SPANISH_VENEZUELA.WE8MSWIN1252

Turkish	TURKISH_TURKEY.TR8MSWIN1254
Ukrainian	UKRAINIAN_UKRAINE.CL8MSWIN1251
Vietnamese	VIETNAMESE_VIETNAM.VN8MSWIN1258

Top of the Document

List of common NLS_LANG settings used in the Command Prompt (DOS box)

Note: this is the correct setting for the DOS BOX SQL*Plus version, (sqlplus.exe/ plus80.exe / plus33.exe)

Operating System Locale	**Oracle Client character set (3rd part of NLS_LANG)**
Arabic	AR8ASMO8X
Catalan	WE8PC850
Chinese (PRC)	ZHS16GBK
Chinese (Taiwan)	ZHT16MSWIN950
Czech	EE8PC852
Danish	WE8PC850
Dutch	WE8PC850
English (United Kingdom)	WE8PC850
English (United States)	US8PC437
Finnish	WE8PC850
French	WE8PC850
German	WE8PC850
Greek	EL8PC737
Hebrew	IW8PC1507
Hungarian	EE8PC852
Italian	WE8PC850
Japanese	JA16SJIS
Korean	KO16MSWIN949
Norwegian	WE8PC850
Polish	EE8PC852
Portuguese	WE8PC850

Romanian	EE8PC852
Russian	RU8PC866
Slovak	EE8PC852
Slovenian	EE8PC852
Spanish	WE8PC850
Swedish	WE8PC850
Turkish	TR8PC857

Top of the Document

Other Frequently asked questions regarding NLS_LANG

What does the LANGUAGE component of the NLS_LANG parameter control?

The language component of the NLS_LANG parameter controls the operation of a subset of globalization support features. It 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.

What does the TERRITORY component of the NLS_LANG parameter control?

The territory component of the NLS_LANG parameter controls the operation of a subset of globalization support features. It 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.

Top of the Document

How to see what's really stored in the database?

To find the real numeric value for a character stored in the database use the dump command:

The syntax of the function call is:

DUMP(<value> [, <format> [, <offset> [, <length>]]])

where:

value – is the value to be displayed

format – is a number which describes the format in which bytes of the value are to be displayed: 8 – means octal, 10 – means decimal, 16 – means hexadecimal; other values between 0 and 16 mean decimal; values greater then 16 are a little confusing and mean: print bytes as ASCII characters if they correspond to printable ASCII codes, print them as "^x" if they correspond to ASCII control codes and print them in hexadecimal otherwise; adding 1000 to the format number will add character set information for the character data type values to the return value offset – is the offset of the first byte of the value to display; negative values mean counting from the end length – is the number of bytes to display. So for example,

SQL> SELECT DUMP(col,1016)FROM table;

Typ=1 Len=39 CharacterSet=UTF8: 227,131,143,227,131,170

returns the value of a column consisting of 3 Japanese characters in UTF8 encoding . For example the 1st char is 227(*255)+131. You will probably need to convert this to UCS2 to verify the codepoint value with the Unicode Standard codepage.

select dump(bank_name, 1016) from eonia.bank_h;

Where is the Character Conversion Done?

Normally conversion is done at client side for performance reasons. This is true from Version 8.0.4 onwards. If the database is using a character set not known by the client then the conversion is done at server side. This is true from Version 8.1.6 onwards.

Windows SQL*Plus is not showing all my extended characters?

You see black squares instead of the characters you probably don't have the right font defined for your codepage. A font is a collection of glyphs (from "hieroglyphs") that share common appearance (typeface, character size). A font is used by the operating system to convert a numeric value into a graphical representation on screen. A font does not necessarily contain a graphical representation for all numeric values defined in the code page you are using. That's why you sometimes get black squares on the screen if you change fonts and the new font has no representation for a certain symbol.

The Windows "Character Set Map" utility can be used to see which glyphs are parts of a certain font.

On Windows 2000 and XP:

Start -> Run...

Type "charmap", and click "ok".

I get a question mark or inverted question mark when selecting back just inserted characters?

When characters are converted between the client and the database character set, or vice versa, the character should exist in both. If it does not exist in the character set being converted to (the destination) then a replacement character is used. Some character sets have specific replacement characters defined when translating from other specific character sets but where this is not done a default replacement character, such as a ?, is used. Conversion from a replacement character back to the original character is not possible.

Is iSQL*Plus the only UTF8/Unicode enabled client we support?

On Windows OS, yes, On Unix OS's, no. All the database utilties, including Import, Export, SQL*Loader, SQL*Plus, can act as a UTF-8 client if the OS locale is UTF-8 (e.g., en_US.UTF-8 on Linux) and NLS_LANG character set is set to UTF8 or AL32UTF8.

How to check the code points managed by a UNIX Operating System?

To know which code point is generated for a character in a Unix Environment, you can use the "od" command:

```
$ echo "" | od -xc
```

You can also check the character corresponding to a code point using the "echo" command like this:

for Solaris, AIX, HP-UX, TRU64:

```
$echo '\0351'
```

for Linux:

```
$echo -e '\0351'
```

You can use Locale Builder or a printed code page (see links below) to verify that your native code page and NLS_LANG setting properly correspond.   If there is any ambiguity then use the command above to get the values for more than one character. For printed code page:

http://www.unicode.org
http://www.iso.org
http://czyborra.com/charsets/iso8859.html

Top of the Document

What about command line tools like SQL*Loader, Import, Export, utilities?

Typically the NLS_LANG needs to match the MS-DOS OEM code page that could be retrieved by typing chcp in a Command Prompt:

```
C:\> chcp

Active code page: 437
```

```
C:\> set NLS_LANG=american_america.US8PC437
```

For tools like SQL*Loader you can temporarily change the NLS_LANG to the character set of the FILE you are loading. An alternative to changing NLS_LANG is to specify the character set of the data in the datafile using the characterset keyword in the .ctl file. In that case, SQL*Loader will interpret the data in the datafile as that character set regardless of the client character set setting of NLS_LANG. Here is an example .ctl file for utf16. This example ships in the demo area:

```
– Copyright (c) 2001 by Oracle Corporation
–  NAME
–   ulcase11.ctl – Load Data in the Unicode Character Set UTF-16
–  DESCRIPTION
–   Loads data in the Unicode character set UTF-16. The data is in little
–   Endean byte order. This means that depending on whether SQL*Loader is
–   running on a little Endean or a big Endean system, it will have to
–   byte swap the UTF-16 character data as necessary. This load uses
–   character length semantics, the default for the character set UTF-16.
–
–   This case study is modeled after case study 3 (ulcase3), which loads
–   variable length delimited (terminated and enclosed) data.
–
–  RETURNS
–
–  NOTES
–   None
–  MODIFIED   (MM/DD/YY)
–   rpfau    02/06/01  – Merged rpfau_sqlldr_add_case_study_11
–   rpfau    01/30/01 -  Creation
–

LOAD DATA
CHARACTERSET utf16
BYTEORDER little
INFILE ulcase11.dat
REPLACE

INTO TABLE EMP
FIELDS TERMINATED BY X’002c’ OPTIONALLY ENCLOSED BY X’0022′
(empno integer external (5), ename, job, mgr,
 hiredate DATE(20) "DD-Month-YYYY",
 sal, comm,
 deptno   CHAR(5) TERMINATED BY ":",
 projno,
 loadseq  SEQUENCE(MAX,1) )
```

In Oracle9*i* the Export utility always exports user data, including Unicode data, in the character set of the database. The Import utility automatically converts the data to the character set of the target database.

In Oracle8*i* the Export utility exports user data converting them from the database character set to the character set of the NLS_LANG of the Export session. The Import utility first converts the data to the character set of the NLS_LANG of the Import session and then converts them to the character set of the target database. Care must be taken that the character set of the NLS_LANG for Export and Import sessions contain all characters to be migrated. This character set is usually chosen to be either the source database or the target database character set and it is usually the same for both Export and Import sessions. This choice is recommended especially with multibyte character sets, which pose some restrictions on export files. The

Oracle8*i* conversions to and from the NLS_LANG character set happen in Oracle9*i* for DDL statements contained in the Export file.

Top of the Document

What about database links?

The NLS_LANG on the server (or client) has no influence on character set conversion through a database link. Oracle will do the conversion from the character set of the source database to the character set of the target database (or reverse).

What about Multiple Homes on Windows?

There is nothing special with NLS_LANG and the multiple homes on Windows. The parameter taken into account is the one specified in the ORACLE_HOME registry key used by the executable. If the NLS_LANG is set in the environment, it takes precedence over the value in the registry and is used for ALL Oracle_Homes on the server/client.

The NLS_LANG can be found in these registry keys:

HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE

or

HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\HOMEx

Is there an Oracle Unicode Client on Windows?

On Windows there are two kinds of tools / applications:

1) A fully Unicode enabled applications which accepts Unicode codepoints and which can render them. It’s the application that needs to deal with the Unicode. Windows provides the Unicode API but the GUI system itself is NOT Unicode "by nature".

A fully Unicode application can only show one glyph for a given Unicode code point. So there is NO confusion possible here, this application will need to use a full Unicode font. If you have a full Unicode application, then you need to set the NLS_LANG to UTF8.

Note that there are currently not many applications like this and if not explicitly stated by the vendor it's most likely an ANSI application. So don't set the NLS_LANG to UTF8 if you are not sure!

The only Unicode capable client that is included in the Oracle database is iSQL*Plus.

2) Standard ANSI application (like sqlplusw.exe) cannot use Unicode code points. So the Unicode code point stored in the database needs to be CONVERTED to an ANSI code point based on the correct setting of the NLS_LANG. This allows Oracle to map the Unicode code point to the character set of the client. If the Unicode code point does not have a mapping to the character set of the client then a replacement character is used.

What is a Character set or Code Page?

A character set is just an agreement on what numeric value a symbol has. A computer does not know 'A' or 'B ', it only knows the (binary) numeric value for that symbol, defined in the character set used by its Operating System (OS) or in hardware (firmware) for terminals. A computer can only manipulate numbers, which is why there is a need for character sets. An example is 'ASCII', an old 7 bit character set, 'ROMAN8' a 8 bit character set on UNIX or 'UTF8' a multibyte character set.

A code page is the name for the Windows/DOS encoding schemes, for Oracle NLS you can consider it the same as a character set. You also have to distinguish between a FONT and a character set/codepage. A font is used by the OS to convert a numeric value into a graphical 'print' on screen. The Wingdings Font on Windows is the best example of a font where an 'A' is NOT shown as an 'A' on screen, but for the OS the numeric value represents an 'A'. So you don't SEE it as an 'A', but for Windows it's an 'A' and will be saved (or used) as an 'A'.

To better understand the explanation above, just open MS Word, choose the Wingdings Font, type your name (you will see symbols) and save this as html, if you open the html file with Notepad you will see that in the <style> section the fonts are declared and lower in the <body> section you will find your name in plain text but with style='font-family: Wingdings' attribute. If you open it in Internet Explorer or Netscape, you will again see the Wingdings symbols. It's the presentation that changes, not the data itself.

It's also possible that you don't see with a particular font, all the symbols defined in the codepage you are using, just because the creator of the FONT did not include a graphical representation for all the symbols in that font. That's why you sometimes get black squares on the screen if you change fonts. On Windows you can use the 'Character Map' tool to see all the symbols defined in a font.

Why Are There Different Character sets?

Two main reasons:

Historically vendors have defined different ‘character sets’ for their hardware and software, mainly because there were no official standards.

New character sets have been defined to support new languages. With an 8 bit character set, you are limited in the number of symbols you can support so there are different character sets for different written languages.

What is the difference between 7 bit, 8 bit and Unicode Character sets?

A 7 bit character set only knows 128 symbols (2^7)

An 8 bit character set knows 256 symbols (2^8)

Unicode (UTF-8) is a multibyte character set. Unicode has the capability to define over a million characters. For more information on Unicode see the white paper Oracle Unicode Database Support.

How to choose the right database character set?

A basic consideration for choosing a character set is to make sure it can handle any language that needs to be supported immediately and in the indeterminate future. Another overlooked consideration is to think about what applications and technologies you may want to utilize or interact with the database. Use locale builder (from Oracle Database 9*i* onwards) to view what characters are defined for a particular Oracle character set.

Choosing Unicode as the database character set ensures a strong foundation for whatever is built into and on top of the database.*Oracle recommends using Unicode for all new system deployment. Migrating legacy systems to Unicode is also recommended. Deploying your systems today in Unicode offers many advantages in usability, compatibility, and extensibility. Oracle’s comprehensive support allows you to deploy high-performing systems faster and more easily while leveraging the true power of Unicode. Even if you don’t need to support multilingual data today or have any requirement for Unicode, it is still likely to be the best choice for a new system in the long run and will ultimately save you time and money and give you competitive advantages.*

Top of the Document

AL32UTF8/UTF8 (Unicode) Database Character Set Implications

May 15, 2008 · 4 Comments

Oracle's recommendation that if your environment (clients and servers) consists entirely of Oracle 9i or higher, use AL32UTF8 as NLS_CHARACTERSET, otherwise use UTF8.

For the time being do NOT use expdp/impdp (Data Pump) when going to UTF8 or another multibyte characterset on ALL 10g versions including 10.1.0.5 and 10.2.0.3. This also includes 11.1.0.6. It will provoke data corruption unless patch 5874989 is applied. This is caused by the impdp Bug 5874989. The older import/export executables work fine and should be used.

If you are using 10.1.0.5, 10.2.0.2, 10.2.0.3 or 11.1.0.6 you can request a backport for Bug 5874989 if there hasn't been a patch issued for your platform.

Some Things to Be Aware Of:

For the database side you do not need "Unicode" support from the OS where the database is running on because the Oracle database is running on because the Oracle AL32UTF8 implementation is not depending on OS features.

Server Side Considerations

AL32UTF8 is a varying width characterset, which means that the code for a character can be 1, 2, 3, or 4 bytes long. This is a big difference with charactersets like WE8ISO8859P1 or WE8MSWIN1252.

US7ASCII characters (A-z,a-z,0-1 and ./?,*,# etc…) are in UTF8 1 byte, so for most West European languages the impact is rather limited as only "special" characters like the unprintable ones will use more bytes then in a 8 bit characterset. But if you convert a Cyrillic or Arabic system to AL32UTF8 then the data will take considerable more bytes to store.

This also means that you have to make sure your columns are big enough to store the additional bytes. By default the column size is defined in BYTES and not in CHARACTERS. So a "create table <name> (<column name> VARCHAR2(2000));" means that the column can store 2000 bytes.

From 9i onwards however, it is possible to really define a column with the number of CHARACTERS you want to store. How this works and what the limits and current known problems are will be explained another day.

Codepoints:

There is a common misconception that a character is always the same code, for example the £is often referred to as "code 163″ character. This is not correct! The code itself means nothing if you do not know what characterset you are using. The difference may look small, but it's not. The pound sign for example is indeed 163 (A3 in hexadecimal) in the WE8ISO8859P1 and WE8MSWIN1252 charactersets, but in AL32UTF8 the pound sign is code 49827 (C2 A3 in hex).

So be careful if you use for example the ASCII or CHR(<code>) function, the code for a character using CHR depends on the database characterset! If you do chr(163) in a AL32UTF8 database then this 163 code is an illegal character, as 163 simply does not exist, for the pound you should use chr(49827) in a AL32UTF8 system.

Instead of CHR() you might use Unistr('\<code>'). Unistr() (is a 9i new feature) always works on every characterset that can display the character.

Objects and User Names are Max 30 Bytes not Characters:

Identifiers of database objects are max 30 bytes long. Names must be from 1 to 30 bytes long with these exceptions:

- Names of databases are limited to 8 bytes.
- Names of database links can be as long as 128 bytes.

Oracle strongly suggests never to use non-US7ASCII names for a database or a database link. The following SQL statement will return all objects having a non-US7ASCII name:

select object_name from dba_objects where object_name <> convert(object_name,'US7ASCII');

A username can be a maximum of 30 bytes long. The name can only contain characters from your database characterset and must follow the rules described in the section "Schema Object Naming Rules". Oracle recommends that the user name contain at least one single-byte character regardless of whether the database character set also contains multibyte characters. The following SQL statement will return all users having a non-US7ASCII name:

select username from dba_users where username <> convert(username, 'US7ASCII');

Using CHAR semantics is for the moment not supported in the SYS schema and that's where the database object and user names are stored. If you have objects or users with non-US7ASCII names that take more than 30 bytes in UTF8 there is no alternative besides renaming the affected objects or user to use a name that will occupy no more than 30 bytes.

The password for users can contain only single-byte characters from your database character set regardless of whether the character set also contains multi-byte characters.

This means that in a (AL32)UTF8 database the user password can only contain US7ASCII characters as they are the only single-byte characters in UTF8. This may provoke a problem, if you migrate from (for example) a CL8MSWIN1251 database then your users can use Cyrillic in their passwords seen in CL8MSWIN1251 cyrillic i single byte, in UTF8 it is not. Note that passwords are stored in a hashed way will NOT be seen in csscan. You will need to reset for those clients the password to US7ASCII string. This restriction has been lifted in 11g, there you can use multi-byte characters as a password string. Please note that they need to be updated in 11g before they use the new 11g hashing system.

Other Things to Watch Out For:

Make sure you use a correct NLS_LANG setting when using export/import when exporting or importing to or from a UTF8 database. Using DBMS_LOB.LOADFROMFILE may have some implications when used in a UTF8 database. When using SQLLDR make sure you define the correct characterset of the "file" when you load. String functions work with characters not byte (length, like, substr ...). There are of course exceptions like lengthB, substrB and instrB who explicitly deal with bytes. Since you never know the exact length of a string in BYTES in a UTF8 environment, operations based on the BYTE length should be avoided, unless you have a compelling reason to use them.

If you do not use CHAR semantics then the column size will be tripled (a CHAR 20 byte will show up as 60 Bytes in the AL32UTF8 db). If you really need to use BYTE you can use the _keep_remote_column_size=true at the MV side, but be aware that this will provoke ORA-1401 or ORA-12899 as the data WILL expand when it contains non-US7ASCII characters. Avoid the use of a mixture of BYTE and CHAR semantics in the same table and _keep_remote_column_size=true is NOT compatible with using CHAR semantics.

The Client Side:

Most people think that the NLS_LANG should be UTF8 or AL32UTF8 because they are connecting to a AL32UTF8 database. This is not necessarily true, the NLS_LANG has in fact no relation with the database characterset. It's purpose is to let oracle know what the client characterset is, so that Oracle can do the needed conversion.

Please make sure that the difference between a client who can't connect to a Unicode database (which is any 8.0 and up client for a UTF8 database and any 9i+ client for AL32UTF8 database) and a "real" Unicode client. A "real" Unicode client means a client who can display/insert all characters known by Unicode without the need to recompile or change the OS settings.

Configuring Your Client To Be a Unicode Client on UNIX:

To hae a Unix Unicode client you need to configure your Unix environment first to use UTF8, then you have to check your telnet software to be able to use Unicode and then (as the last step) you can set the NLS_LANG environment variable equal to AMERICAN_AMERICA.AL32UTF8 and start SQL*Plus.

Configuring Your Client To Be a Unicode Client on Windows:

On Windows you cannot use sqlplusw.exe or sqlplus.exe as a Unicode / UTF8 client. Using sqlplusw.exe with NLS_LANG set to AL32UTF8 is totally incorrect. There are 2 Oracle provided Unicode clients you can use on Windows: iSQL*Plus and Oracle SQL Developer. If you want to use / write a Unicode application on Windows then the application should be specifically written to use the Windows Unicode API, setting NLS_LANG to AMERICAN_AMERICA.AL32UTF8 is not enough. You should consult the application vendor or the vendor of the development environment to see how to properly code in Unicode.

The popular tool TOAD is NOT able to run against (AL32)UTF8 databases. Toad does not support databases with a Unicode character set such as UTF8 or AL16UTF16. The International version of Toad is the only version of Toad that does support (AL32)UTF8.

All Oracle 9i and up clients are compatible with a AL32UTF8 database, even if they use a non-AL32UTF8 NLS_LANG setting:

Note: You can use sqlplusw.exe on a Western European / US Windows Client (using a NLS_LANG set to AMERICAN_AMERICA.WE8MSWIN1252, which is the correct value for a Western European US Windows system to connect to a AL32UTF8 database. However, the SQL*Plus client will only be able to insert / view Western European characters. If another user using SQL*Plus on a correctly configured Chinese Windows System inserts data then this will not be visible in the Western European SQL*Plus client. If you try to update Chinese data using the Western European client this will also fail.

Some links where we can find different character sets and what their individual characters mean

http://msdn.microsoft.com/en-us/goglobal/cc305145.aspx

http://9stmaryrd.com/tools/character-encodings/table

http://arjudba.blogspot.com/2009/03/difference-between-we8iso8859p1-and.html

http://www.av8n.com/computer/utf/font-chart.html

```
SQL>set serveroutput on
declare
i number;
begin
for i in 0..255 loop
 declare
    ch varchar2(1);
 begin
    ch := chr(i);
    if  convert( ch, 'WE8ISO8859P1', 'WE8ISO8859P15') != ch
    then
      dbms_output.put_line('Difference- Decimal:'|| i ||' Hexa:'||
to_char(i,'XXXX'));
    end if;
 end;
end loop;
end;
```

The source data is 88591 we use ETL informatica it is having 11 charcter set of data since 88591 is 1 byte of data it gives samecode for different charcterset for example it is having x(code)=man(japan),women(korean) it raises ambiguity in database for japan and korean language , and now at present target is also 88591 the same "x" is sent to target but here there own languge is set in their system , if koren guy sees it he will undersatnd it is women and if japan guy sees he will understand as man but now we are going to make target database into utf-8 and informatica run in unicode mode here the source system is in 88591 as i have told u earlier it may generate same code for some charcterset now when we load it into target which is utf-8 here it generates unique code for different charcterset but we need to identify the end user reqirement and give him yhe exact data.

example
If end user is korean in earlier case it is x but now utf-8 generates unique code so we need to tell to informatca before loading in to target there it supports all charcaterset and give unique code for each charcter set.

My intention:-
==============
We will be deciding at the time of running sessions or one time conversion yo flat file to utf-8 and then to target.

Although I know problem may seem hazy.Lets make it a lil bit clear before putting the solution.

A database named ABCD is defined to only support one character set(ISO-8859-1), data is getting populated here with data from multiple character sets like sjis,big5, GB2312 etc. We accept that the ordering of the data is according to ISO-8859/1

Slowly as time passes by ABCD will have text data in multiple different languages in multiple different character sets and later it becomes tough for identifying which language and character set the text belongs to. The UTF8 encoding of UNICODE, which keeps any current text in USASCII unchanged (the vast majority of our text data), but stores data from other character sets in 2, 3, or 4 byte units.

Now there is a requiremwnt to transfer data from ABCD toanother database named EFGH which is in UNICODE.So we need to be able to identify the character set of every text string.Lets assume we have identified that also.

Question is that how to perform that data transfer through INFA7.1

Solution:-
Thats can be done by INFA.Just keep following things in mind.

1/
Check what is the type of your source database character set (select * from nls_database_parameters) 2/ Check what is the type of your target database character set(select * from nls_database_parameters) 3/ Check what data movement has been set for Informatica Server which you are to assign in your workflow.
(Go to the config file you use to pass while starting informatica server in UNIX)

Eg.
# Determine one of the two server data movement modes: UNICODE or ASCII.
# If not specified, ASCII data movement mode is assumed.
# ASCII:-PowerServer processes single byte character and does not perform codepage
#conversion

```
#UNICODE:-Processes 2 bytes for a character.Enforce codepage validation

DataMovementMode=Unicode
```
/**
Set it Unicode,only then the end users will have full data else while there will be corrupt data.
If you are resetting,after resetting restart the Informatica Server service.
**/
4/
If you have set all those things right,then there is nothing to worry.Users should must see Data as per their locale.
5/
You may face some LM_ error while loading data through INFA.In that case revert me back with error log portion like
```
/**************************************************************************************************
MAPPING> CMN_1569 Server Mode: [UNICODE] CMN_1570 Server Codepage: [ISO
MAPPING> 8859-1 Western European]
**************************************************************************************************/
```

```
$ grep Baden 20101231_PR_D_P_D_4.csv | head -1 | cut -f37 -d "," | od -c

0000000  "  L  a  n  d  e  s  b  a  n  k     B  a  d  e

0000020  n  -  W 303 274  r  t  t  e  m  b  e  r  g  "  \n

0000040

$
```

and 303 274 is the octal representation of "ü" in Unicode.

A 8859 "ü" looks like

```
$ cat ue.txt | od -c

0000000  B  a  d  e  n  -  W 374  r  t  t  e  m  b  e  r

0000020  g

0000021

$
```

Where 374 is the octal representation of "ü" in 8859.

Rating: 10.0/**10** (1 vote cast)

Rating: **+3** (from 3 votes)

- Bookmark on Delicious
- Digg this post
- Recommend on Facebook
- share via Reddit
- Share with Stumblers
- Tweet about it
- Subscribe to the comments on this post

(Visited 39,303 times, 52 visits today)

This entry was posted in interesting advanced dba articles and tagged new dba, new oracle dba, new oracle dba character set, oracle character set, oracle dba. Bookmark the permalink.

- follow:
- RSS