ORACLE® **White Paper**

Designing Multilingual Applications Using Developer/2000 and Oracle7

November 1995

Summary
This paper illustrates how Developer/2000 and Oracle7 can be used to develop multilingual applications. Design techniques are described that enable a single application to process multilingual data while adapting its end-user interface to the language and national conventions of each end-user. Such a ‘single-source’ solution significantly reduces the cost of development and maintenance of applications deployed globally. Emphasis is given to describing design techniques that minimise the complexity of application design. Also, the process of translating Developer/2000-based applications is explained.

Introduction

For an organization operating in a multi-lingual and/or multi-country environment, a multilingual application can present many advantages. Only one application need be developed, providing all the features needed to configure itself to the specific language and country conventions of each end-user. These features address the following basic requirements:

- adaption to language and data-format conventions
- catering for different character encoding schemes
- processing multilingual data
- displaying end-user interface text in the user’s language

Each of these features is discussed in the context of how they can be met by a Developer/2000-based application that takes advantage of the National Language Support Architecture provided by Oracle7. Where relevant, future planned enhancements to these capabilities are also mentioned.

National Language Support Architecture

A multilingual application requires an underlying National Language Support Architecture capable of supporting all languages and character encoding schemes used on computer systems. Also, such an Architecture has to provide an extensive range of National Language Support (NLS) features such as date and number formatting and sorting of character data. Oracle7 provides such an Architecture, capable of supporting all ‘groups’ of languages:

- European Languages: English, French, German, Greek, Russian, ...
  - alphabets encoded with single-byte encoding schemes

- Semitic Languages: Arabic, Hebrew, ...
  - alphabets encoded with single-byte encoding schemes
  - dual writing direction

- Asian Languages: Chinese, Japanese, Korean ...

- alphabets encoded with multi-byte encoding schemes

Language-dependent features operate at runtime according to the NLS environment specified for each end-user. By taking advantage of this capability, the design of a multilingual application can be greatly simplified compared to ‘hard-coding’ such features as part of the application design itself. Also, the same Oracle product versions can used at all locations where the application is installed.

Language Adaption

A user’s NLS environment is specified by defining the parameter NLS_LANG using the syntax *<language>_<territory>.<character encoding>,* for example on a UNIX system:

```
$setenv NLS_LANG French_France.WE8ISO8859P1
```

This controls, in addition to other features, the default formatting of numbers and dates, for example:

```
select ename, hiredate, round(sal/12,2) sal
from   emp;

ENAME      HIREDATE   SAL
Müller     01/04/89   3795,83
Höscht     10/05/90   2933,33
Hélène     01/11/91   4066,67
```

Another user could specify a different NLS_LANG value, for example:

```
$setenv NLS_LANG American_America.WE8ISO8859P1
```

in which case, the default formatting would change to:

```
select ename, hiredate, round(sal/12,2) sal
from   emp;

ENAME      HIREDATE   SAL
Müller     01-APR-89  3795.83
Höscht     10-MAY-90  2933.33
Hélène     01-NOV-91  4066.67
```

This ‘automatic’ adaption is transparent to the application design in the sense that it is transparent to the SQL statements used. If specific formats are required, language-independent format masks can be used to achieve the same degree of transparent adaption, for example:

	NLS_LANG	
	American_America	French_France
to_char(<date>,'DD/Mon/YY')	7/Feb/93	17/Fev/93
to_char(<num>,'99G999D99')	74,195.83	74.195,83

The number of characters in a default date format can change for different <*territory*> conventions, hence the application design does need to ensure that they are correctly displayed for the range of <*territory*> values the application supports. This is not normally an issue with number formats, since only the characters used for the radix and group separator change.

A multilingual application must avoid the use of language-dependent date and number literals in SQL statements, for example:

```
create view staff as select * from emp
where  hiredate    > '1-JAN-89'
and    sal         > '69999.00';
```

Such a view evaluates correctly only if the user's NLS environment is compatible with the date and number formats. Such SQL statements should instead use language-independent formats, for example:

```
create view staff as select * from emp
where  hiredate    > TO_DATE ('1-JAN-89', 'DD-MON-YY',
                     'nls_date_language = American')
and    sal         > TO_NUMBER ('69999.00', '99G999D99',
                     'nls_numeric_characters = ''.,''');
```

This view evaluates correctly independent of the NLS environment in force for the user session.

In the 7.2 Server release, additional calendars will be supported, for example the Japanese Imperial and Arabic Hijrah. Where these use fixed formats, they will override the Gregorian format mask specified. Hence, while it will be possible to build an application that adapts to different calendars for input and display of dates, differences in length will need to be taken into account in the application design.

Adaption also applies to sorting sequences. Each <*language*> specifies a default sort sequence used in ORDER BY queries, for example:

```
$setenv NLS_LANG German            $setenv NLS_LANG Swedish

select letter from letters         select letter from letters
order by letter;                   order by letters;

LETTER                             LETTER
a                                  a
ä                                  b
b                                  z
z                                  ä
```

Again, the change in sort sequence is transparent to the application design. In addition to setting the NLS environment prior to logon, it can also be changed during a session using the ALTER SESSION statement, for example:

```
     alter session set    NLS_DATE_LANGUAGE = German
                          NLS_DATE_FORMAT = 'DD.MON.YY'
                          NLS_NUMERIC_CHARACTERS = '.,';
```

In the 7.2 Server release, all NLS parameters can be specified in addition to NLS_LANG to provide greater flexibility in defining the desired combination of language and territory conventions prior to logon.

Language Customization

Linguistic and cultural conventions are not ‘standardised’. The example above illustrates the different sorting conventions in German and Swedish. Even where the same language is used, usage can be different, for example French in France and Canada. To cater for this multiplicity, Oracle7 provides a wide range of pre-defined language and territory conventions and sorting sequences. The latter can also cater for special cases relating to digraphs, or ‘double’ characters, such as the German ‘ß’ and Spanish ‘ch’ and ‘ll’.

A further complexity is the multitude of different character encoding schemes used on computer systems. These have been devised both by computer manufacturers and standards organizations, mostly based on either the 7-bit ASCII standard or IBM's 8-bit EBCDIC. Multi-byte schemes are used for Asian languages since these use thousands of characters. In order for data processing operations, for example converting a character from lowercase to uppercase, testing whether a character is a valid alphanumeric, to give the correct result, the Oracle software must know which encoding scheme the data is in, and the detailed specification of that encoding scheme. Oracle7 includes a pre-defined set such specifications, identified by an acronym in the general form:

<language grouping><number of bits><originator>

For example, WE8ISO8859P1 refers to the ISO 8859/1 8-bit encoding scheme that supports many West European languages.

The NLS_LANG parameter is used to specify the character encoding scheme being used for input and output of data by the application. SQL functions used for character manipulation (for example SUBSTR) operate per-character, not per byte. (Oracle7 provides some additional SQL functions - for example SUBSTRB that do operate per-byte). Hence, a single application can be designed to operate with both single-byte and multi-byte encoding schemes. However, column length specifications in table definitions refer to bytes, not characters. For example,

create table emp (ename varchar2(10));

allows at most 10 bytes to be stored in the column *ename*. For a multi-byte encoding scheme, the number of characters would be less than 10 depending on the nature of the encoding scheme. The choice of appropriate column lengths does need to be taken into account for applications that span both single and multi-byte encoding schemes. Such considerations also apply to the choice of display field lengths, for example in a Forms application.

A side-effect of case conversion of digraphs is that the number of characters in the result can differ from that in the source. For example, the uppercase of the German ‘ß’ is ‘SS’. To avoid compatibility problems with existing applications, UPPER, LOWER and INITCAP do not cater for such digraphs - for example UPPER(‘ß’) returns ‘ß’. Equivalent functions - NLS_UPPER, NLS_LOWER and NLS_INITCAP - are provided that do, but must be used explicitly in an application if required.

Semitic languages are supported in both Developer/2000 and Oracle7. Developer/2000 specifically provides support for controlling:

- Layout direction - allows displaying items with labels at the right of the item and also enables correct placement of check boxes and radio buttons.
- Reading order - allows specifying whether the text direction is left->right or right->left.
- Alignment - allows the point of origin to be upper right or upper left.
- Initial keyboard state - allows specifying whether local or Roman characters will be produced during data entry.

Support for additional conventions and encoding schemes are added to Oracle7 releases on a continuing basis. Before Oracle7.2, such data specifications were linked into the Oracle software, and because of their large number (more than 150 encoding schemes are currently supported overall). Since the 7.2 Server release, data specifications are stored externally and are loadable at runtime. This has eliminated the need for linked-in data and provides a simple mechanism to enable on-site customization of such specifications, for example to modify a specific sorting sequence or add support for an encoding scheme.

Client/Server Character Encoding Schemes

In general, two character encoding schemes are involved in a user session, the encoding scheme being used by the input/output device (for example character mode terminal, or PC running Windows) and the encoding scheme used to store data in the Oracle7 database. The latter is defined when a database is created with CREATE DATABASE, the former with the parameter NLS_LANG. The two encoding schemes can be different, and different for different clients. Any required conversion of data between encoding schemes is done by the Oracle software, and is transparent to the application design.

Where different encoding schemes are used, It is normally essential to ensure that they are compatible in that all support the repertoire of characters capable of being handled by the application. If a character in the source data is not defined in the target encoding scheme, replacement characters are used. These are defined as part of the specifications for the target encoding scheme, and can define replacements on a per-character basis if relevant. If not, a default character (usually a question mark) is used. To ensure complete conversion, the target encoding scheme must contain all characters in the source data. This is discussed further in the section below.

Conversion is always possible between any two single-byte encoding schemes, between two multi-byte schemes, and between a single-byte and a multi-byte scheme for a limited set of combinations. However, in converting data between different multi-byte schemes, string lengths may change, and this may impact application design.

Conversion also applies to data transferred between two servers where the encoding schemes used in the two databases are different. A further issue is to ensure that the NLS environment operating on behalf of a specific user session is propagated to any remote servers that the user session connects to (for example in a distributed query). Such synchronization (for language and territory conventions) is done automatically when such connections are made, and is transparent to the application design.

Multilingual Data

The range of languages that can be supported by a multilingual application is inherently limited to the range of languages supported by the character encoding scheme being used. For data input and output, this is determined by the input/output device, and its hardware/software must enable an appropriate encoding scheme to be used according to the application requirements. For example, for West European languages many UNIX systems support ISO 8859/1. Many computer manufacturers also support the use of their own schemes that were devised to support the same language group, for example DEC Multinational, HP Roman8, IBM PC Code Page 850 and EBCDIC Code Page 500. However, such single-byte schemes can only support a limited range of characters, and other schemes must be used, for example, for East European languages and non-Latin based languages such as Greek and Russian. Multi-byte schemes can support many more characters, but were principally designed to support a specific Asian language (for example Japanese, Korean, Traditional or Simplified Chinese) and also only support a limited range of languages. All encoding schemes support the basic Latin alphabet (a to z).

Where the required range of languages falls within the scope of a single encoding scheme, the choice of encoding schemes should be straightforward. The encoding scheme used by the client application (which is made known to the Oracle software via NLS_LANG) is that determined by the input/output device. If this is the same for all users, it is also the natural choice for the encoding scheme to be used for the database. Different users can use different encoding schemes, in which the database encoding scheme should be chosen as the most relevant (for example to minimise overall data conversion). The only restriction in choice of database encoding scheme is that an EBCDIC-based encoding scheme can not be used for a database on an ASCII-based server, and vice versa. As discussed above, where different encoding schemes are used, it is normally essential to ensure that their character repertoires (the list of characters defined in a specific encoding scheme) are compatible to ensure that replacement characters are not used during conversion.

Ensuring this is much more of a problem where the language group is wider than that which is supported within one encoding scheme (for example West European languages plus Greek). In such cases, a multiple database solution may be required, since only one encoding scheme can be defined per database. For this example, one database could be used for West European character data and a second for Greek character data. The application design would have to ensure that only data capable of being displayed (and updated) on the specific input/output device can be accessed by a user session. In some cases, use of a custom encoding scheme may be preferable to achieve a situation where a single encoding scheme can be used. Such a scheme however also has to be supported by the input/output device.

UNICODE

A ‘complete’ solution to such limitations is the aim of the UNICODE character encoding scheme (now a subset of the ISO 10646 standard). UNICODE defines all characters used in all languages in a single encoding scheme. It is fixed length with 2 bytes being used for every character, even a - z. Hence it is not compatible with ASCII or EBCDIC. ‘Conventional’ multi-byte schemes are variable length, generally 1 byte being used for the Latin alphabet, two bytes (or more) for Asian characters. The fixed byte encoding of UNICODE (referred to as UCS-2) has the disadvantage that Latin character data requires more storage space that with ‘conventional; multi-byte schemes, but has the advantage that all bytes are quaranteed to be 2-bytes in length, which enables the use of more efficient coding techniques for processing such data compared to those required for ‘conventional’ multi-byte schemes.

In additional to being non-ASCII/EBCDIC compatible, UCS-2 also has the property that 0 bytes can occur within character strings. Code written to use null-terminated strings can therefore not cater for UCS-2. In general, existing software, even if it can cater for variable-length multi-byte encoding schemes, has to be modified to cater for UCS-2. Hence, the process of providing support for UCS-2 can require significant development work.

X/Open have defined an alternative encoding scheme for the UNICODE character repertoire, referred to as UTF-8. This uses a variable-length, multi-byte, ASCII-compatible encoding which can be supported more easily by software that already caters for 'conventional' multi-byte schemes, at least on ASCII-based platforms. The 7.2 Server release will provide support for UTF-8, enabling this to be used as the encoding scheme for a database. Hence, a single database can then store character data for any group of languages.

However, this in itself does not overcome the limitation of what can be input and displayed by the client application. Character data can always be converted from any 'conventional' single or multi-byte encoding scheme to UTF-8 (or UCS-2), but not vice versa. The application design still needs to ensure that only data compatible with the input/output device being used is accessed by the user session. If the input/output device can itself use UTF-8, or at least that subset that the application is required to support, then this limitation is also removed. DEVELOPER/2000 will also provide support for UTF-8 in a future release, enabling this to be used where the input/output devise also supports UTF-8. Support for UCS-2 is also being planned for future releases of both Server and DEVELOPER/2000.

Translating Applications

An important consideration in maximising the ease-of-use of an application, and hence minimising end-user training costs, is to display the end-user interface of the application (for example, screen displays, error and help messages) in the native language of the end-user. In general, such text originates from two sources, the Oracle software and the application design itself. To support a multilingual application, multiple language versions of both sources of text need to be available and the language used selected at runtime. Such multilingual text capability is provided by Oracle7 and Developer/2000. Translations for Oracle products are distributed by Oracle itself for an increasing range of products and languages. Developer/2000 supports multilingual application text in that multiple versions can be created for a single application. Translation is part of the application development process.

Oracle Forms

In the case of Oracle Forms, a single application definition (FMB/MMB) file can contain multiple versions of application text (for example trigger messages, boilerplate text, default values, menu items, button labels). Separate runtime (FMX/MMX) files are then generated for each language. If it is required to provide users with a choice of language, the application design needs to provide some sort of menu-driven front-end to identify and execute the relevant FMX file.

Within the FMB file, different versions of the application text can be stored using different character encoding schemes if this is necessary. An FMX file is generated with the text in the encoding scheme used in the FMB file for that language. If the user's input/output device uses a different encoding scheme (as indicated by NLS_LANG) text is automatically converted to that encoding scheme.

Text displayed to the end-user from a PL/SQL procedure using the MESSAGE() function is not catered for automatically. In these cases, an application-specific solution must be used, for example using a message table in the database itself.

Translation Manager

Translation Manager is a tool designed to manage and perform the translation of Developer/2000-based applications. It is a component of Developer/2000, and optimized for translation of the type of text typically used in Developer/2000 applications. It can also be used to translate Oracle products themselves in the case where the required translation is not available from Oracle itself.

Translation Manager does not provide automated machine-translation, a technology not well suited to the type of 'unstructured' text used in applications. Rather, Translation Manager provides features to minimise the cost and time taken to do translation, and to maximise the productivity of the translator. For example, Translation Manager maintains a translation memory to store previous translations. This provides the capability of automated upgrade of translations to a new version of a translated application to re-use previous translations that are still valid. The translation memory can also be applied to new applications in a search/replace mode, again to maximise re-use of existing valid translations.

Source text is loaded from an application source file into Translation Manager's translation memory, and the translator then uses a simple split-screen edit window to perform the required translation work. This provides a unified translation environment, with an easy-to-learn interface, and does not require the translator to have any knowledge or training in the application development environment itself. Also, it ensures that the application design is completely isolated from the translation process, to ensure that a translator can make no inadvertent changes to the application design itself. Once translation is complete, Translation Manager is used to merge the translated text back into the application source file.

Conclusions

Multilingual applications have evident benefits compared to maintaining separate versions of applications for each language. Also, implementing multilingual capability into the application design itself has formerly required complex design techniques. The aim of this paper has been to show that Oracle7 and Developer/2000 can be used to build multilingual applications with minimal increase in design complexity compared with that required for a monolingual application. This is possible by taking advantage of the key features of transparent runtime adaption of language-dependent behaviour, and multilingual application text capability.

However, application design does need to take into account the limitations imposed by input/output devices in the range of characters they are able to create and display. However the emergence of UNICODE may well provide a solution to this limitation as it becomes more widely supported by operating system and application development software, and by input/output devices themselves.

ORACLE ®

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
USA

Worldwide Inquiries:
Ph   415.506.7000
Fax 415.506.7200

Author: NLS Product Management
Date: November 1995