Building Global Internet Applications with the Oracle Globalization Development Kit

An Oracle White Paper
November 2003
Building Global Internet Applications with the Oracle Globalization Development Kit

Introduction ...........................................................................................................3
What is the Oracle Globalization Development Kit (GDK)? .............................4
Internet application models ....................................................................................5
  Multiple instances of monolingual Internet applications ............................5
  Single instance of a multilingual Internet application ...............................5
Developing a global Internet application ............................................................6
  Determining the user locale preference and synchronizing it with the
  application ...........................................................................................................6
    Supported locale sources .................................................................................6
  Detecting the user locale ....................................................................................7
  Determining the locale sources .......................................................................8
  Defining the supported application locales in the GDK ............................8
Specifying the character set for the input and output of web content .................9
Staging and managing translated content .........................................................11
  Static files ..........................................................................................................11
    Strings in the jsp/java servlets .....................................................................12
Getting consistent globalization operations between the database and
the application ......................................................................................................13
  Oracle locale mapping .....................................................................................14
  Oracle locale information .................................................................................14
  Oracle character set conversion .....................................................................14
  Oracle date, number, and monetary formatting ..........................................15
  Oracle binary and linguistic sorts ...................................................................16
  Oracle language and character set detection ...............................................16
  Oracle translated locale and time zone names .............................................17
Conclusion ...........................................................................................................17
INTRODUCTION

The Internet, by nature is called the World-Wide-Web because it addresses a global market. Internet applications have the infrastructure and capability to support users with different language and cultural requirements. A truly global Internet application needs to support data exchange in a wide range of character sets, and the user interface must present data in a format that matches the local convention of the user’s language and country.

Designing and developing a global application can be a daunting task even for the most experienced developers. They are required to have an understanding of the fundamental concepts of implementing a global architecture into the design of their application. Developers need to design and write code capable of supporting multiple end users with different character sets and locale preferences. The application must be able to interact and synchronize with the locale model in the database server. The complexity associated with the different globalization concepts, can make building and supporting a global Internet application quite a challenge for many companies.

Oracle Database 10g introduces the Oracle Globalization Development Kit (GDK), it provides a framework for accelerating the development of globalized Internet applications. The GDK is a toolkit that simplifies the development process, and reduces the cost of developing Internet applications that will be used to support a global environment. The key component of the GDK is a set of Java APIs, which provide Oracle application developers with the framework to develop globalized Internet applications using the best globalization practices and features designed by Oracle. Although J2EE platform already provides a strong foundation for building globalized applications, its globalization functionalities and behaviors can be quite different from Oracle’s functionalities. The GDK complements the existing globalization features in J2EE. The GDK provides synchronization of locale sensitive operations between the middle-tier Java application and the database server. The GDK also contains a suite of PL/SQL packages that provide additional globalization functionalities for applications written in PL/SQL.
This paper focuses on the Java GDK framework. It illustrates the simplicity and the benefits of building global Internet applications with the GDK.

**WHAT IS THE ORACLE GLOBALIZATION DEVELOPMENT KIT (GDK)?**

The functionalities offered by the GDK can be divided into two distinct areas, *The GDK application framework* and *the GDK Java API*.

- **The GDK application framework** encapsulates the complexity of globalization programming, such as determining user locale, maintaining locale persistency and processing locale information. It supports the detection of the user-preferred locale, the re-writing of URLs to locate content for the different locales, and provides a single application configuration file to control the globalization operation of the user application. The *GDK application framework* provides the development support for the J2EE middle tier to hide the complexity of synchronizing globalization operations across tiers.

- **The GDK Java API** provides development support in the middle tier to provide consistent globalization operations as provided in the database server. It extends Oracle Globalization Support features to the application server, by allowing applications to perform globalization logic such as Oracle date/number/monetary formatting and Oracle binary & linguistic sorting in the middle tier. It provides mappings between Oracle and Java locales, character set conversions between Oracle and Java character sets,

![Diagram of the Globalization Development Kit (GDK)](image)
and it also offers locale sensitive information such as the common languages, linguistic sorts and the local time zones supported by a given territory etc.

The GDK Java API also offers developers advanced globalization features, such as Language and Character Set detection technology.

INTERNET APPLICATION MODELS

There are two basic models for multilingual Internet application deployment. The two models are:

**Multiple instances of monolingual Internet applications**

Internet applications that can support only one locale in a single application are classified as monolingual applications. A locale refers to a language and the region in which the language is spoken. For example, the primary language of the United States and Great Britain is English. However, the two territories have different currencies and different conventions for date formats. Therefore, the United States and Great Britain are considered to be 2 different locales.

Using this model, supporting a new locale usually requires the introduction of a separate application instance, and sometimes even a new middle-tier server. This is the traditional and the most common application deployment model.

Typically, the application was created with the intention of serving customers with only one locale need. When a request to support a new set of locale comes in, developers will manually replace the hard coded text strings to the new language and spawn a separate code branch to maintain the different locale versions. This model is manageable only if the number of supported locales is very small.

**Single instance of a multilingual Internet application**

Internet applications that support multiple locales simultaneously in a single application are classified as multilingual applications. This deployment architecture is suitable for customers who want to support several locales in an Internet application simultaneously. Multilingual applications are deployed to the middle-tier with a single application server configuration that works for all locales. The major advantages of this model over the monolingual model are:

- It simplifies the deployment configuration and hence reduces the cost of maintenance.
- Enhancements to the application can be applied once and be available for all locales.
- Providing support to new locales is simple. No new application instances are required to support the new locales.
- Testing the application across different locales can be done in a single testing environment.
Multilingual content can be supported within the same instance of the application. For example, a web page containing multiple language data can be shown.

The disadvantage of deploying multilingual Internet applications is the long development cycle, due to the extra complexities involved in the design and the development of the applications. To support multiple locales in a single application, there are many design considerations. The application must be able to dynamically detect the preferred locale of the users. It needs to adapt to the locale by constructing HTML pages in the language and cultural conventions of the user. Text data entered by users or those retrieved from the database must be processed in Unicode, so that data in any language can be supported. Application messages and UI text should be externalized and staged together with their translated counterparts, so that the localized files can be retrieved based on the locale of the user. These are all development challenges that can be simplified by using the GDK.

The GDK contains the basic building blocks for writing multilingual Internet applications, however many of the features from the GDK Java API can also be applicable when building monolingual applications.

DEVELOPING A GLOBAL INTERNET APPLICATION

Building an Internet application that supports multiple locales requires good globalization programming practices. Here are some of the key development focus areas:

- Determining the user locale preference and synchronizing it with the application.
- Specifying the character set for the input and output of web content.
- Staging and managing translated content.
- Getting consistent globalization operations between the application and the database.

The GDK provides facilities to reduce the complexities in dealing with the above globalization development challenges.

Determining the user locale preference and synchronizing it with the application

Supported locale sources

To be able to determine the user’s preferred locale is the first step in supporting a multilingual application. In a web environment, several locale sources are available. The locale detection offered by the J2EE platform is primitive, Java supports locale detection via the HTTP language preference only.
The **GDK** provides support for multiple locale sources. The description of each predefined locale source is as follows:

<table>
<thead>
<tr>
<th>Locale Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP language preference</td>
<td>Locale or locales included in the HTTP protocol as a value of Accept-Language. This is set at the web browser level. A locale fallback operation is required if the browser locale is not supported by the application.</td>
</tr>
<tr>
<td>User input locale</td>
<td>Locale specified by the user from a menu or a parameter in the HTTP protocol.</td>
</tr>
<tr>
<td>Locale preference from database</td>
<td>Locale preference stored in the database as part of the user profiles.</td>
</tr>
<tr>
<td>Application default locale</td>
<td>A locale defined in the <strong>GDK</strong> application configuration file. This locale is defined as the default locale for the application. Typically, this is used as a fallback locale when the other locale sources are not available.</td>
</tr>
</tbody>
</table>

*Table 1 Locale sources supported by the GDK*

The **GDK** provides support for predefined locale sources; such as user input locale, HTTP language preference, a locale preference in the database, and the application default locale, seamlessly. Custom locale sources, such as locale preference from the LDAP server can be easily integrated into the **GDK** framework by creating database access objects.

The **GDK** application configuration file `gdkapp.xml`, is a key component of the **GDK** application framework. It dictates the globalization behaviors and the properties of the application. It contains locale mapping tables, character sets of web contents and globalization parameters for the configuration of the application. One application configuration file is required per J2EE application consuming **GDK**, and it should be placed in the `/WEB-INF` directory of the J2EE environment of the application. The application administrator can modify the application configuration file to change the globalization behavior in the application, without the need to change the programs and to recompile them.

**Detecting the user locale**

`ServletHelper.getLocalizerInstance(HttpServletRequest)` that is associated with the current locale. All locale related information is obtained from the Localizer object. This object includes functionalities of the following classes: `OraDateFormat`, `OraNumberFormat`, `OraCollator`, `OraLocaleInfo`, and `LocaleMapper`.

In general, if you require more than the functionality of `request.getLocale`, then using the Localizer within the **GDK** application framework is recommended. The **GDK** `Localizer` hides the complexity of dealing with globalization specific logic, therefore it minimizes the effort required to make your applications global ready.
Determining the locale sources

The Locale detection logic described above is dependent on the locale type defined in the GDK application configuration file. The following example shows the locale determination rule section of the application configuration file. It indicates that the user preferred locale could be determined from either the LDAP server or from the HTTP Accept-Language header.

```
<locale-determine-rule>
  <locale-source>LDAPUserSchema</locale-source>
  <locale-source>oracle.i18n.servlet.localesource.HTTPAcceptLanguage</locale-source>
</locale-determine-rule>
```

The user locale is detected based on the order listed in the application configuration file. For example, if the user is authenticated and the user locale preference is stored in the LDAP database; LDAPUserSchema connects to the database and retrieve the user locale preference. When the user is anonymous, HTTPAcceptLanguage returns the user language preference of the web browser.

Defining the supported application locales in the GDK

The number of locales and the names of the locales that an application needs to support are based on the business requirements and the targeted user base. The names of the application supported locales are registered in the application configuration file. The following example shows the application locales section of the application configuration file. It indicates that the application supports de-* (German for all countries), ja (Japanese) and en-US (English for the US), with English defined as the default application locale.

```
<application-locales>
  <locale>de-*</locale>
  <locale ja</locale>
  <locale default="yes">en-US</locale>
</application-locales>
```

During the locale detection phrase, the GDK verifies whether the user locale is defined as a supported locale in the application configuration file. The locale verification algorithm used in the GDK is as follows:

1. Check if the detected locale is included in the supported locale list. If yes, use this locale as the user locale.
2. If there is a variant in the detected locale, remove the variant and repeat step 1. For example, de-DE-EURO has the variant EURO, remove the variant and make it de-DE.
3. If the locale includes a country code, remove the country code and repeat step 1. For example, ja-JP has the country code JP, remove the country code and it becomes ja.
4. If the detected locale is not defined in the application supported locale list,
use the default application locale as the user locale.
By performing steps 2 and 3, the GDK will be able to support additional users with the same language requirements but have different locale settings than those defined in the application configuration file. Using the above example, the application will be able to support users with the following locale preferences de-AT (German in Austria), de-CH (German in Switzerland) and de-LU (German in Luxembourg) also.

The above locale fallback mechanism in the GDK is similar to that of the Java resource bundle. The exception is that the GDK is not affected by the default locale of the Java VM, this is because the application default locale is used during the locale fallback operations.

**Specifying the character set for the input and output of web content**
The character set or encoding of an HTML page is a critical piece of information to a web browser and an Internet application. The browser needs to interpret this information so that it can use the correct fonts and character set mappings for displaying pages. The Internet application needs to know so it can safely process input data from a HTML form based on the specified character set. The page encoding can be translated as the character set used for the locale to which an Internet application is serving.

In order to correctly specify the page encoding for HTML pages, application developers must:
1. Determine the desired page encoding for a given locale.
2. Specify the corresponding encoding name on each HTML page.

Using the GDK, the above steps can be eliminated. No additional application code is required to tag the pages with the correct character set information. The character set information can be specified in the GDK application configuration file. At runtime, the GDK automatically sets the character sets for the request and response objects.

The GDK supports the following three models for specifying the character sets of the HTML pages:

- A single local character set dedicated to the whole application - this is usually more appropriate for a monolingual Internet application. However, depending on the language coverage of the character set, it may be possible to support more than one language using this model. For example, using ISO-8859-1, most Western European languages can be served.

- Use Unicode UTF-8 for all contents regardless of the language - this is suitable for a multilingual application that uses Unicode for deployment.

- Use the native character set for each language - this is appropriate for a multilingual application that uses a default character set mapping for each locale. For example, English contents are served in ISO-8859-1 and
Japanese contents are represented in Shift_JIS. This model is sometimes required for applications that need to support different character sets based on the user locales. One example can be for mobile applications that lack Unicode font support or web browsers, which cannot fully support Unicode.

The character set is specified in the application configuration file. The following is an example of setting UTF-8 as the character set for all the application pages.

```xml
<page-charset>UTF-8</page-charset>
```

The page charset information is used by ServletRequestWrapper and ServletResponseWrapper, which set the proper character set for the request object. It is also used by ContentType for output when instantiated. If the `page-charset` is set to AUTO-CHARSET then the character set is automatically assigned by the GDK for the current user locale.

```xml
<page-charset>AUTO-CHARSET</page-charset>
```

The default locale character mappings are derived from the LocaleMapper class, which provides the default IANA character set to a locale in the GDK.

The following table lists the mappings between the common ISO locales and their IANA character sets.

<table>
<thead>
<tr>
<th>ISO locale</th>
<th>NLS_LANGUAGE</th>
<th>NLS_TERRITORY</th>
<th>IANA Character Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>ar-SA</td>
<td>ARABIC</td>
<td>SAUDI ARABIA</td>
<td>WINDOWS-1256</td>
</tr>
<tr>
<td>de-DE</td>
<td>GERMAN</td>
<td>GERMANY</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>en-US</td>
<td>AMERICAN</td>
<td>AMERICA</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>en-GB</td>
<td>ENGLISH</td>
<td>UNITED KINGDOM</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>el</td>
<td>GREEK</td>
<td>GREECE</td>
<td>WINDOWS-1253</td>
</tr>
<tr>
<td>es-ES</td>
<td>SPANISH</td>
<td>SPAIN</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>fr</td>
<td>FRENCH</td>
<td>FRANCE</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>fr-CA</td>
<td>CANADIAN FRENCH</td>
<td>CANADA</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>iw</td>
<td>HEBREW</td>
<td>ISRAEL</td>
<td>WINDOWS-1255</td>
</tr>
<tr>
<td>ko</td>
<td>KOREAN</td>
<td>KOREA</td>
<td>EUC-KR</td>
</tr>
<tr>
<td>ja</td>
<td>JAPANESE</td>
<td>JAPAN</td>
<td>SHIFT_JIS</td>
</tr>
<tr>
<td>it</td>
<td>ITALIAN</td>
<td>ITALY</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>pt</td>
<td>PORTUGUESE</td>
<td>PORTUGAL</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>pt-BR</td>
<td>BRAZILIAN PORTUGUESE</td>
<td>BRAZIL</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>tr</td>
<td>TURKISH</td>
<td>TURKEY</td>
<td>WINDOWS-1254</td>
</tr>
<tr>
<td>nl</td>
<td>DUTCH</td>
<td>THE NETHERLANDS</td>
<td>WINDOWS-1252</td>
</tr>
<tr>
<td>zh</td>
<td>SIMPLIFIED CHINESE</td>
<td>CHINA</td>
<td>GBK</td>
</tr>
<tr>
<td>zh-TW</td>
<td>TRADITIONAL CHINESE</td>
<td>TAIWAN</td>
<td>BIG5</td>
</tr>
</tbody>
</table>

*Table 2 Common ISO Locales to IANA character set mapping*
To provide further flexibility, the locale to character set mapping in the GDK is customizable. To override the default mapping defined in the GDK, a locale-to-character-set mapping table can be specified in the application configuration file.

\[
\begin{align*}
\text{<locale-charset-maps>}
\quad \text{<locale-charset>}
\quad \text{<locale>ja</locale><charset>EUC-JP</charset>}
\quad \text{</locale-charset>}
\quad \text{</locale-charset-maps>}
\end{align*}
\]

The above example shows that for the locale ja (Japanese), the application will replace the default character set from SHIFT_JIS to EUC-JP.

### Staging and managing translated content

#### Static files

Static files such as HTMLs, GIFs, and XML are readily translatable. To differentiate the languages of the translated files, the files are usually staged either with different file names or in different directories. For example, the French version of index.html can be staged either as index_fr.html or fr/index.html. This information is then used to construct the URLs, hard coded into the application.

By using the `rewriteURL()` method, the GDK will handle the logic to locate the translated files from the corresponding language directories. The `ServletHelper_rewriteURL()` method rewrites a URL based on the rules specified in the application configuration file. This method is used to determine the correct location where the localized content is staged. The following is an example of the JSP code:

\[
\begin{align*}
&\text{<img src="%=ServletHelper_rewriteURL("image/logo.jpg", request)"} >
\quad \text{<a href="%=ServletHelper_rewriteURL("html/welcome.html", request)"} >
\end{align*}
\]

The URL rewrite definitions are defined in the **GDK** application configuration file:

\[
\begin{align*}
\text{<url-rewrite-rule fallback="yes">}
\quad \text{<pattern>(.*)/([^/]+)$</pattern>}
\quad \text{<result>$1/$A/$2</result>}
\quad \text{</url-rewrite-rule>}
\end{align*}
\]

The pattern and the result sections defined in the rewrite rule follow the regular expression conventions. The symbols $L, $C, $V, and $A are used to represent the ISO 639 language code, ISO 3166 country code, the locale variant string, and the full locale string as derived from java.util.Locale respectively. $1 to $9 are used to represent the matched sub strings in the parameter passed by the ServletHelper class.
Using the above example, if the current user locale is ja, then the `rewriteURL()` method will change the URLs for the `logo.jpg` image file to `image/ja/logo.jpg`, and `welcome.html` to `html/ja/welcome.html`.

**Strings in the JSP/Java servlets**

ResourceBundle is the technique that allows access to localized contents at runtime in Java. Translatable strings within Java Servlets are externalized into Java resource bundles so that these resource bundles can be translated independently into the different languages. The translated resource bundles carry the same base class names as the English bundles, using the Java locale name as the suffix.

Typically, to retrieve translated data from the ResourceBundle, the `getBundle()` method must be invoked for every request.

```jsp
<% Locale user_locale=request.getLocale();
    ResourceBundle rb=ResourceBundle.getBundle("resource",user_locale); %>
<%=
    rb.getString("Welcome") %>
```

The GDK simplifies the retrieval of text strings from the resource bundles. The `Localizer.getMessage()` method is a wrapper to ResourceBundle.

```jsp
<% Localizer.getMessage("Welcome") %>
```

Instead of specifying the base class name to the `getBundle()` method in your application, you can specify the name in the application configuration file. The GDK automatically instantiates when a translated text string is requested. Using the GDK approach, developers are not required to hardcode the base class name into their applications. If the resource bundle name needs to be changed; the application administrator just needs to update one line in the application configuration file.

The following example shows the message-bundles section of the application configuration file.

```xml
<message-bundles>
    <resource-bundle>resource</resource-bundle>
    <resource-bundle name="resource2">resource2</resource-bundle>
</message-bundles>
```

The default message bundle for the application will be used as if the attribute name is not specified. For applications that need to support more than one resource bundle, the additional resource bundle names and their aliases can be added into the application configuration file. The non-default resource bundle name is a parameter to the `getMessage()` method. Here is an example of calling the `getMessage()` method using different resource bundles.
String translatedMessage1 =
Localizer.getMessage("Hello");
String translatedMessage2 =
localizer.getMessage("World", "resource2");

Both ServletHelper.rewriteURL() and
Localizer.getMessage() methods perform locale fallback operations in
the case where the translation files for the user locale are not available. For
example, if the online help files are not available for the locale es_MX (Spanish
for Mexico), but the es (Spanish for Spain) are; then the methods will select the
Spanish translated files as the substitute.

GETTING CONSISTENT GLOBALIZATION OPERATIONS BETWEEN THE
DATABASE AND THE APPLICATION

Java's globalization functionalities and behaviors can be very different to those
offered by the Oracle database. In fact, even the terminology and the architecture
between the two are not the same. Oracle uses territory and language settings
inside the database, whereas Java has the concept of a locale, which contains
both the language and the region combined together. Another example is that
Java supports a set of locales and character sets that are different from Oracle.
There are more than 30 Oracle territories and languages, and over 100 Oracle
code sets in Oracle Database 10g, which are not supported in Java JDK
1.4.2.

The inconsistency between the two can create confusion for users, especially
when their application contains data that are formatted based on two different
conventions. E.g. Dates that are retrieved from the database are formatted using
Oracle conventions, (such as number and date formatting and linguistic sort
ordering), but the static application data are typically formatted using Java locale
conventions.

The GDK extends Oracle's database globalization features to the application
server. By allowing applications to perform globalization operations such as
Oracle date/number, formatting and character set conversion in the middle tier.
The GDK allows developers to eliminate expensive programming logic from the
database, hence improving the overall application performance by reducing
unnecessary network traffic between the application tier and the database server.

Here are some of the Oracle Globalization features, which are available in the
GDK:

- Oracle Locale Mapping
- Oracle Locale Information
- Oracle Character Set Conversion
- Oracle Date, Number, and Monetary Formatting
- Oracle Binary and Linguistic Sorts
• Oracle Language and Character Set detection
• Oracle Translated locale and time zone names

**Oracle locale mapping**
The GDK provides the `LocaleMapper` class that maps equivalent locales and character sets between Java, IANA, ISO and Oracle. A Java application may get locale specifications from clients that are specified in Oracle’s locale name or IANA character set name. The Java application needs to be able to map to an equivalent Java locale or Java encoding before it knows how to process the information correctly.
The following is an example of the `LocaleMapper` class:

```java
// From Java Locale to Oracle language and Oracle territory
Locale locale = new Locale("it", "IT");
String oraLang = LocaleMapper.getOraLanguage(locale);
String oraTerr = LocaleMapper.getOraTerritory(locale);

// From Oracle language and Oracle territory to Java Locale
locale =
LocaleMapper.getJavaLocale("AMERICAN","AMERICA");
locale = LocaleMapper.getJavaLocale("TRADITIONAL CHINESE", "");
```

**Oracle locale information**
Oracle locale definitions, which include languages, territories, linguistic sorts, and character sets, are exposed in the GDK. `OraLocaleInfo` is an Oracle locale class that includes language, territory, and collator objects. It provides a method for applications to retrieve a collection of locale related objects for a given locale, for example, a list of the common Oracle linguistic sorts available for sorting data in a given language, the local time zones defined for a given territory, or the common languages used in a particular territory.

The following is an example of using the `OraLocaleInfo` class to return a list of time zones applicable to the United States:

```java
// Local Time Zones for a given Territory
OraLocaleInfo oloc =
OraLocaleInfo.getInstance("AMERICAN", "AMERICA");
TimeZone[] loctz = oloc.getLocalTimeZones();
```

**Oracle character set conversion**
Although Java JDK is already equipped with APIs that can perform conversions for many of the standard character sets, they do not support Oracle-specific character sets or Oracle's user-defined character sets.
The GDK provides implicit support for Oracle’s character sets by utilizing the character set plug-in feature, supported via the `java.nio.charset` package in JDK 1.4. The GDK character set conversion supports all Oracle character sets including user-defined characters sets. It can be used by Java applications to properly convert to and from Java’s internal character set, UTF-16.

To avoid potential conflicts with Java’s own character sets, all Oracle character set names have an `x-oracle-` prefix for all implicit usage through Java’s API.

The following is an example of Oracle Character set conversion:

```java
// Converts the Japanese character "three" from UTF-16 to JA16SJIS
String str = "\u4e09";
byte[] barr = str.getBytes("X-ORACLE-JA16SJIS");
```

**Oracle date, number, and monetary formatting**

The GDK provides formatting classes that support date, number, and monetary formats using Oracle conventions for Java applications in the `oracle.i18n.text` package. New cultural sensitive formats introduced in Oracle Database 10g, such as the new short and long date formats, number and monetary formats for different locales, are also available in these classes.

The following are examples of Oracle date, Oracle number, and Oracle monetary formatting:

```java
// Obtain the current date and time in the default Oracle LONG format for
// the locale de_DE (German_Germany)
Locale locale = new Locale("de", "DE");
OraDateFormat odf =
    OraDateFormat.getDateTimeInstance(OraDateFormat.LONG, locale);

// Obtain the numeric value 1234567.89 using the default number format
// for the Locale en_IN (English in India)
locale = new Locale("en", "IN");
OraNumberFormat onf =
    OraNumberFormat.getNumberInstance(locale);
String nm = onf.format(new Double(1234567.89));

// Obtain the monetary value 1234567.89 using the default currency
// format for the Locale en_US (American_America)
locale = new Locale("en", "US");
onf = OraNumberFormat.getCurrencyInstance(locale);
nm = onf.format(new Double(1234567.89));
```
Oracle binary and linguistic sorts

Oracle provides support for binary, monolingual, and multilingual linguistic sorts inside the database. In Oracle Database 10g, these sorts have been enhanced to provide case-insensitive and accent-insensitive sorting and searching capabilities. By using the `OraCollator` class, the GDK enables Java applications to sort and search for information based on the latest Oracle binary and linguistic sorting features, including the new case insensitive and accent insensitive options.

The following is an example of string comparison using the XGERMAN monolingual linguistic sort:

```java
// Compares strings using XGERMAN
private static String s1 = "abcSS";
private static String s2 = "abc\u00df";
String cname = "XGERMAN";
OraCollator ocol = OraCollator.getInstance(cname);
int c = ocol.compare(s1, s2);
```

Oracle language and character set detection

The Oracle Language and Character Set Detection library in the GDK provides a high performance, statistically based engine for determining the character set and language for unspecified text. It can automatically identify the language, character set pairs for most of the common languages including Asian, and Arabic scripts. With each text, the language and character set detection engine sets up a series of probabilities, each probability corresponding to a language and character set pair. The most probable pair statistically identifies the dominant language and character set.

The `LCSDetector` class detects the language and character set of a byte array, a char array, a string and an `InputStream`. It can take the entire input for sampling or only portions of the input for sampling, when the length of both the offset and the length are supplied. For each input, up to 3 potential language and character set pairs can be returned by the `LCSDetector` class. They are always ranked in sequence, with the pair with the highest probability returned first.

Please refer to the Oracle Database 10g Globalization Support Guide for a list of supported language and character sets pairs.

The following is an example of using the `LCSDetector` class to enable Language and Character Set detection:

```java
// This example shows how to use the LCSDetector class to detect the language and character set of a byte array
int          offset = 0;
```
LCSDetector led = new LCSDetector();

while ( true )
{
    bytes_read = led.detect(byte_input, offset, 1024);
    if ( bytes_read == -1 )
        break;
    offset += bytes_read;
}
LCSDResultSet res = led.getResult();

**Oracle translated locale and time zone names**

All the Oracle language names, territory names, character set names, linguistic sort names and time zone names have been translated into 27 languages including English. They are readily available for inclusion into the user applications, and they provide consistency for the display names across user applications in different languages.

The Oracle translated names are useful for presentation in UI text and for drop down selection boxes. For example, a native French user would prefer to select from a drop down list of time zone names displayed in French than in English.

**OraDisplayLocaleInfo** is a translation utility class that provides the translations of locale and attributes.

The following is an example of using **OraDisplayLocaleInfo** to return a list of time zones supported in Canada, using the French translation names:

```java
OraLocaleInfo oloc = OraLocaleInfo.getInstance("CANADIAN FRENCH", "CANADA");
OraDisplayLocaleInfo odloc = OraDisplayLocaleInfo.getInstance(oloc);
TimeZone[] loctzs = oloc.getLocaleTimeZones();
String[] disptz = new string[loctzs.length];
for (int i=0; i<loctzs.length; ++i)
{
    disptz[i] = odloc.getDisplayTimeZone(loctzs[i]);
    ...
}
```

**CONCLUSION**

The Oracle Globalization Development Kit (GDK) provides a framework for accelerating the development of globalized Internet applications. The GDK complements existing features in Java and J2EE while removing the complexity of developing global applications. It brings important Oracle globalization features to the middle tier, and handles the compatibility between Java and the Oracle database server seamlessly.

Oracle Database 10g and the GDK provide the ideal Internet platform to build and deploy global Internet applications.