21
Globalization Support

The Oracle Java Database Connectivity (JDBC) drivers provide globalization support, formerly known as National Language Support (NLS). Globalization support enables you to retrieve data or insert data into a database in any character set that Oracle supports. If the clients and the server use different character sets, then the driver provides the support to perform the conversions between the database character set and the client character set.

This chapter contains the following sections:

> **Note:**
>
> - Starting from Oracle Database 10*g*, the `NLS_LANG` variable is no longer part of the JDBC globalization mechanism. The JDBC driver does not check NLS environment. So, setting it has no effect.
> - The JDBC server-side internal driver provides complete globalization support and does not require any globalization extension files.
> - JDBC 4.0 includes methods for reading and writing national character set values. You should use these methods when using JSE 6 or later.

Related Topics

- Oracle Character Data Types Support
- *Oracle Database Globalization Support Guide*

21.1 About Providing Globalization Support

The basic Java Archive (JAR) files, `ojdbc11.jar` and `ojdbc17.jar`, contain all the necessary classes to provide complete globalization support.

This support includes the following:

- Oracle character sets for `CHAR`, `VARCHAR`, `LONGVARCHAR`, or `CLOB` data that is not being retrieved or inserted as a data member of an Oracle object or collection type.
- `CHAR` or `VARCHAR` data members of object and collection for the character sets `US7ASCII`, `WE8DEC`, `WE8ISO8859P1`, `WE8MSWIN1252`, and `UTF8`.

To use any other character sets in `CHAR` or `VARCHAR` data members of objects or collections, you must include `orai18n.jar` in the `CLASSPATH` environment variable:

ORACLE_HOME`/jlib/orai18n.jar`

> **Note:**
>
> Previous releases depended on the `nls_charset12.zip` file. This file is now obsolete.

Compressing orai18n.jar

The `orai18n.jar` file contains many important character set and globalization support files. You can reduce the size of `orai18n.jar` using the built-in customization tool, as follows:

```
java -jar orai18n.jar -custom-charsets-jar [jar/zip_filename] -charset characterset_name
[characterset_name ...]
```

For example, if you want to create a custom character set file, `custom_orai18n_ja.jar`, that includes the JA16SJIS and JA16EUC character sets, then issue the following command:

```
$ java -jar orai18n.jar -custom-charsets-jar custom_orai18n_ja.jar -charset JA16SJIS
JA16EUC
```

The output of the command is as follows:

```
Added Character set : JA16SJIS
Added Character set : JA16EUC
```

If you do not specify a file name for your custom JAR/ZIP file, then a file with the name `jdbc_orai18n_cs.jar` is created in the current working directory. Also, for your custom JAR/ZIP file, you cannot specify a name that starts with `orai18n`.

If any invalid or unsupported character set name is specified in the command, then no output JAR/ZIP file will be created. If the custom JAR/ZIP file exists, then the file will not be updated or removed.

The custom character set JAR/ZIP does not accept any command. However, it prints the version information and the command that was used to generate the JAR/ZIP file. For example, you have `jdbc_orai18n_cs.zip`, the command that displays the information and the displayed information is as follows:

```
$ java -jar jdbc_orai18n_cs.jar
Oracle Globalization Development Kit - 12.1.X.X.X Release
This custom character set jar/zip file was created with the following command:
java -jar orai18n.jar -custom-charsets-jar jdbc_orai18n_cs.jar -charset WE8ISO8859P15
```

The limitation to the number of character sets that can be specified depends on that of the shell or command prompt of the operating system. It is certified that all supported character sets can be specified with the command.

Note:

If you are using a custom character set, then you need to perform the following so that JDBC supports the custom character set:

1. After creating the `.nlt` and `.nlb` files as part of the process of creating a custom character set, create `.glb` files for the newly created character set and also for the `lx0boot.nlt` file using the following command:

   ```
   java -classpath $ORACLE_HOME/jlib/orai18n.jar:$ORACLE_HOME/lib/
   xmlparserv2.jar Ginstall -[add | a] <NLT_file_name>
   ```

2. Add the generated files and `$ORACLE_HOME/jlib/orai18n-mappings.jar` into the `classpath` environment variable while executing the JDBC code that connects to the database with the custom character set.

21.2 NCHAR, NVARCHAR2, NCLOB and the defaultNChar Property

By default, the `oracle.jdbc.OraclePreparedStatement` interface treats the data type of all the columns in the same way as they are encoded in the database character set. However, since Oracle Database 10*g*, if you set the value of `oracle.jdbc.defaultNChar` system property to `true`, then JDBC treats all character columns as being national-language.

The default value of `defaultNChar` is false. If the value of `defaultNChar` is false, then you must call the `setFormOfUse(<column_Index>, OraclePreparedStatement.FORM_NCHAR)` method for those columns that specifically need national-language characters. For example:

```
PreparedStatement pstmt =
conn.prepareStatement("insert into TEST values(?,?,?)");
pstmt.setFormOfUse(1, OraclePreparedStatement.FORM_NCHAR);
pstmt.setString(1, myUnicodeString1); // NCHAR column
pstmt.setFormOfUse(2, OraclePreparedeStatement.FORM_NCHAR);
pstmt.setString(2, myUnicodeString2); // NVARCHAR2 column
```

If you want to set the value of `defaultNChar` to `true`, then specify the following at the command-line:

```
java -Doracle.jdbc.defaultNChar=true myApplication
```

If you prefer, then you can also specify `defaultNChar` as a connection property and access `NCHAR`, `NVARCHAR2`, or `NCLOB` data.

```
Properties props = new Properties();
props.put(OracleConnection.CONNECTION_PROPERTY_DEFAULTNCHAR, "true");
// set URL, username, password, and so on.
...
Connection conn = DriverManager.getConnection(props);
```

If the value of `defaultNChar` is `true`, then you should call the `setFormOfUse(<column_Index>, FORM_CHAR)` for columns that do not need national-language characters. For example:

```
PreparedStatement pstmt =
conn.prepareStatement("insert into TEST values(?,?,?)");
pstmt.setFormOfUse(3, OraclePreparedStatement.FORM_CHAR);
pstmt.setString(3, myString); // CHAR column
```

> **Note:**
>
> If you set the value of `defaultNChar` to `true` and then access `CHAR` columns, then the database will implicitly convert all `CHAR` data into `NCHAR`. This conversion has a substantial performance impact.

> **Note:**
>
> - Always use `java.lang.String` for character data instead of `oracle.sql.CHAR`. `CHAR` is provided only for backward compatibility.
> - You can also use the `setObject` method to access national character set types, but if the `setObject` method is used, then the target data type must be specified as `Types.NCHAR`, `Types.NCLOB`, `Types.NVARCHAR`, or `Types.LONGNVARCHAR`.

> **Note:**
>
> In Oracle Database, SQL strings are converted to the database character set. Therefore you need to keep in mind the following:
>
> - In Oracle Database 10*g* release 1 (10.1) and earlier releases, JDBC drivers do not support any `NCHAR` literal (n'...') containing Unicode characters that are not representable in the database character set. All Unicode characters that are not representable in the database character set get corrupted.
> - If an Oracle Database 10*g* release 2 (10.2) JDBC driver is connected to an Oracle Database 10*g* release 2 (10.2) database server, then all `NCHAR` literals (n'...') are converted to Unicode literals (u'...') and all non-ASCII characters are converted to their corresponding Unicode escape sequence. This is done automatically to prevent data corruption.
> - If an Oracle Database 10*g* release 2 (10.2) JDBC driver is connected to an Oracle Database 10*g* release 1 (10.1) or earlier database server, then `NCHAR` literals (n'...') are not converted and any character that is not representable in the database character set gets corrupted.

21.3 New Methods for National Character Set Type Data in JDK 6

JDBC 4.0 introduces support for the following four additional SQL types to access the national character set types:

- `NCHAR`
- `NVARCHAR`
- `LONGNVARCHAR`
- `NCLOB`

These types are similar to the `CHAR`, `VARCHAR`, `LONGVARCHAR`, and `CLOB` types, except that the values are encoded using the national character set. The JDBC specification uses the `String` class to represent `NCHAR`, `NVARCHAR`, and `LONGNVARCHAR` data, and the `NClob` class to represent `NCLOB` values.

To retrieve a national character value, an application calls one of the following methods:

- `getNString`
- `getNClob`
- `getNCharacterStream`
- `getObject`

> **Note:**
>
> The `getClob` method may be used to return an `NClob` object since `NClob` implements `Clob`.

To specify a value for a parameter marker of national character type, an application calls one of the following methods:

- `setNString`
- `setNCharacterStream`
- `setNClob`
- `setObject`

> **Note:**
>
> You can use the `setFormOfUse` method to specify a national character value in JDK 6. But this practice is discouraged because this method will be deprecated in future release. So, Oracle recommends you to use the methods discussed in this section.

> **See Also:**
>
> If the `setObject` method is used, then the target data type must be specified as `Types.NCHAR`, `Types.NCLOB`, `Types.NVARCHAR`, or `Types.LONGNVARCHAR`.