Date Handling in Oracle Developer

An Oracle White Paper

10/07/99

CONTENTS

0.1 Dates in Oracle Developer

0.1.0.1 A note on the product name

This product was formerly called Developer/2000. The name was changed to Oracle Developer with Release 6. For the sake of simplicity, all releases of the product are referred to by this latter name (Oracle Developer) throughout this paper.

0.1.0.2 What's new in this edition

- Release numbers have been updated.
- The new product name is used.
- A suggestion has been added to also set the date environment variables for the environment in which you will design and compile your form.
- Placing format mask definitions in a PRE-FORM trigger (as previously recommended) may not be adequate in all cases. Solutions are now provided in step 2 of "Suggestions for New Forms Applications." See the expanded material on page -56 and following. This also has modified step 4 of "Steps to Implement the RR Technique" starting on page -63.

0.1.0.3 How this paper is organized

Two types of information are provided here: explanations of how dates are handled in the Oracle Developer products; and some guidance in moving towards Year 2000 compliance in applications created with these products.

The date-handling explanations are also divided into two major sections: first the earlier releases that do not have date-handling enhancements, and then the later releases that do. Specifically:

- Releases 1.2, 1.3.2, 1.5.1, and 1.6.0 of Oracle Developer. Releases in this category do not have the date-handling enhancements available in later releases. The explanations in this section are further subdivided by product:

  - Form Builder (releases 4.5.6, 4.5.7.0 to 4.5.7.16.2, 4.5.8, and 4.5.9)
  - Report Builder (releases 2.5.4.0.8, 2.5.5.2.5, 2.5.5.2.7, 2.5.7.2, and 2.5.7.3)
  - Graphics Builder (release 1.6.x).

- Releases 1.3.3, 1.6.1, 2.1, and 6.0 of Oracle Developer. The date-handling functions in these releases have been enhanced. Material in this section is also further subdivided by product:

  - Form Builder (releases 4.5.7.18.0, 4.5.10, 5.0.6, and 6.0.x)
  - Report Builder (releases 2.5.5.20, 2.5.7.4.3, 2.5.7.5, 3.0.5, and 6.0.x)
  - Graphics Builder (releases 2.5.x and 3.0.x).

  Release 2.0 of Oracle Developer is also discussed in this section. This release is partially enhanced. Forms Release 5.0.5 contains most of the enhancements documented for the other enhanced releases. (The exceptions are noted.) In this release of Oracle Developer, Reports and Graphics contain PL/SQL support for RR and RRRR.

The suggestions for Year 2000 compliance are also divided into two sections by the same release levels:

- Releases 1.2, 1.3.2, 1.5.1, and 1.6.0 of Oracle Developer. (As noted earlier, releases in this category do not have the date-handling enhancements available in later releases.) The suggestions in this section are further subdivided by product:

  - Form Builder (releases 4.5.6, 4.5.7.0 to 4.5.7.16.2, 4.5.8, and 4.5.9)
  - Report Builder (releases 2.5.4.0.8, 2.5.5.2.5, 2.5.5.2.7, 2.5.7.2, and 2.5.7.3)
  - Graphics Builder (release 1.6.x).

- Releases 1.3.3, 1.6.1, 2.1, and 6.0 of Oracle Developer. (As noted earlier, releases in this category have more advanced date-handling functions.) The suggestions in this section are also further subdivided by product:

  - Form Builder (releases 4.5.7.18.0, 4.5.10, 5.0.6, and 6.0.x)
  - Report Builder (releases 2.5.5.20, 2.5.7.4.3, 2.5.7.5, 3.0.5, and 6.0.x)
  - Graphics Builder (releases 2.5.x and 3.0.x)

Other Data Sources:

The descriptions of date handling in this paper assume you are using an Oracle7 or Oracle8 Database Server as your data source. If you are using a non-Oracle data source via ODBC, or using an Oracle Lite or Rdb data source, see the appendix, “Applications that Use Other Data Sources,” on page -75.

0.1.0.4 Overview of dates in Oracle Developer

Dates in a Oracle Developer application come from several sources:

- fetched from the server/database
- entered by the end user
- defined in the application itself

As the end users do their processing, dates move back and forth through the application.

A date as seen by end users is not in the same format as it is within the application, or as it resides in the database. In other words, as dates move through the application, they undergo a translation or conversion.

This conversion process can be fairly complex. It depends on many things:

- the original nature of the date object
- the direction the date object is being moved
- the relevant format masks that will modulate the conversion.

In addition to the format masks the programmer might explicitly specify, the Oracle Developer products use a number of their own internal masks. To add to the complexity, some of these masks can have their values changed by various runtime environment variables set for the application.

Date handling in Oracle Developer, then, includes many factors: varieties of data types, choices made by the application designer, environment owner, and end user, various conversion operations performed internally, plus the interactions between all of these.

Solving existing date problems in your applications, and avoiding future problems, requires understanding this rather involved picture.

0.1.1 Dates in Oracle Developer -- Non-Enhanced Releases

This section covers date handling in Releases 1.2, 1.3.2, 1.5.1, and 1.6.0 of Oracle Developer. Forms, Reports, and Graphics are each discussed in their own section.

Additional Information

If you need a refresher on the specific mask elements in a date format mask, see the section "Date Format Models" in your *Oracle Server SQL Reference* manual.

0.1.1.1 Form Builder, Releases 4.5.6, 4.5.7.0 to 4.5.7.16.2, 4.5.8, and 4.5.9

The handling of dates in these releases of Form Builder depends on several things: on the type of date object involved, on the conversion operation being performed, and especially on which of the many format masks are being used.

Types of date objects

In Forms, a date object is considered to be either a DATE2 object, a DATE4 object, or a DATETIME object. The following table shows this categorization. In this paper, these objects are referred to collectively as date objects.

Table 0–1 Types of date objects in non-enhanced releases

Date object	Type
Item of datatype DATETIME	DATETIME
Item of datatype DATE in Oracle Developer Release 1.2:	
n with length (Maximum Length) of 10 or less	DATE2
n with length (Maximum Length) of 11 or more	DATE4

Table 0–1 Types of date objects in non-enhanced releases

Date object	**Type**
Item of datatype DATE in Oracle Developer Release 1.3 and higher:	
n having a format mask that contains yyyy, YYYY, rrrr, or RRRR	
n having a format mask that does not contain yyyy, YYYY, rrrr, or RRRR	DATE4
n not having a format mask, and of length (Maximum Length) of 9 or less	DATE2
n not having a format mask, and of length (Maximum Length) of 10 or more	DATE2
	DATE4
Parameter (as in :PARAMETER.myparam) of datatype DATE. (Note that there are no DATETIME parameters, and that a parameter's Maximum Length property applies only to CHAR parameters.)	DATE2
LOV column of datatype DATE. (Note that there are no DATETIME LOV columns.)	DATE2
Internal value of system variable CURRENT_DATETIME and EFFECTIVE_DATE	DATETIME

Types of conversion operations

Form Builder performs internal conversion operations in several circumstances:

n when converting a date to a string

n when converting a string to a date

n when converting a user-entered date for display in an error message.

The two tables below list the various conversion operations. They also show which format masks are used. The format masks themselves are explained after the tables.

Table 0–2 Date-to-string operations in non-enhanced releases of Forms

Operation requiring converting a date to a string	Format mask used
Formatting:	
▪ Displaying a value in an item that has a format mask	The item’s format mask
▪ Displaying a value in an item that has no format mask	Output
▪ Displaying a value in a date LOV column	Output
Internal Forms operations:	
▪ Getting :SYSTEM.CURRENT_DATETIME	Builtin
▪ Getting :SYSTEM.EFFECTIVE_DATE	Builtin
▪ Setting a CHAR return item from a date LOV column	Output (but see note 1)
▪ Setting a CHAR item's initial value from $$DATE$$, $$DATETIME$$, $$DBDATE$$, or $$DBDATETIME$$	Output (but see note 1)
▪ GET_ITEM_PROPERTY(item, DATABASE_VALUE)	Output (but see note 1)
▪ GET_ITEM_PROPERTY(item, ITEM_DEFAULT_VALUE)	Output (but see note 1)
▪ GET_ITEM_PROPERTY(item, RANGE_HIGH/LOW)	Output
▪ GET_LIST_ELEMENT_VALUE(item, value)	Builtin
▪ GET_PARAMETER_ATTR(list, key, paramtype, value) for a date parameter defined at design time	Builtin
▪ NAME_IN(variable)	Builtin
PL/SQL V1:	
▪ TO_CHAR(value, format_mask) in PL/SQL	That format_mask (but see note 2)
▪ TO_CHAR(value) in PL/SQL	DD-MON-YY
▪ Implicit conversion in PL/SQL	DD-MON-YY
Database SQL (includes database SQL embedded in PL/SQL):	
▪ TO_CHAR (value, format mask) in database SQL	That format mask
▪ TO_CHAR (value) in database SQL	Database
▪ Implicit conversion in database SQL	Database
▪ Storing a date value returned by database SQL into a Forms CHAR variable	Database
▪ Converting a Forms date variable when it needs to be inserted into or compared against a CHAR column	Database

Table 0–2 Date-to-string operations in non-enhanced releases of Forms

Operation requiring converting a date to a string	Format mask used
PL/SQL debugger:	
n Displaying the value of an item or a parameter in the debugger	Builtin
n Displaying the value of a local PL/SQL variable in the debugger	DD-MON-YY

(Notes follow next table.)

-

Table 0–3 String-to-date operations in non-enhanced releases of Forms

Operation requiring converting a string to a date	Format mask used
Formatting:	
n Extracting a value from an item that has a format mask	The item’s format mask
n Extracting a value from an item that has no format mask	Input
Internal Forms operations:	
n Setting :SYSTEM.EFFECTIVE_DATE	Builtin
n Setting a DATE or DATETIME return item from a CHAR LOV column	Input (but see note 1)
n Setting a DATE or DATETIME item’s initial value from a CHAR variable (using colon notation)	Input (but see note 1)
n ADD_LIST_ELEMENT(item, index, label, value)	Builtin
n SET_PARAMETER_ATTR(list, key, paramtype, value) for a date parameter defined at design time	Builtin
n COPY(value, variable)	Builtin
n DEFAULT_VALUE(value, variable)	Builtin
PL/SQL V1:	
n TO_DATE(value, format_mask) in PL/SQL	That format mask (but see note 2)
n TO_DATE(value) in PL/SQL	DD-MON-YY
n Implicit conversion in PL/SQL	DD-MON-YY

Table 0–3 String-to-date operations in non-enhanced releases of Forms

Operation requiring converting a string to a date	Format mask used
Database SQL (includes database SQL embedded in PL/SQL):	
n TO_DATE(value, format_mask) in database SQL	That format mask
n TO_DATE(value) in database SQL	Database
n Implicit conversion in database SQL	Database
n Storing a CHAR value returned by database SQL into a Forms DATE variable	Database
n Converting a Forms CHAR variable when it needs to be inserted into or compared against a date column	Database
PL/SQL debugger:	
n Setting the value of an item or a parameter in the debugger	Builtin
n Setting the value of a local PL/SQL variable in the debugger	DD-MON-YY

Table notes:

1. In Release 4.5.6 of Form Builder, the Builtin format mask is used instead of the Output format mask.
2. A format mask specified in a PL/SQL TO_CHAR or TO_DATE construct must not contain the RR or RRRR format mask element. However, these elements are permissible in TO_CHAR and TO_DATE constructs in database SQL operations (queries, inserts, updates, deletes).

Types of date format masks

The non-enhanced releases of Form Builder use the following date format masks:

1. User-created date format masks for individual items
2. Builtin date format mask
3. Database date format mask
4. Input date format mask
5. Output date format mask
6. Error date format mask

Each mask is discussed below.

1. Individual items' date format masks

These are set by the programmer or user at design time or runtime, through the Format_Mask property.

Note that there is no requirement that every item have its own individual mask. Indeed, where possible this should be avoided, since the presence of such masks makes internationalization difficult. (Items without individual masks are processed by Form Builder using its general-purpose Input and Output masks.)

2. Builtin date format mask

This mask is used for internal operations within Forms when converting a date value to or from a string that is not potentially visible to the end user.

The value of the Builtin date format mask is chosen based on the type of the date object being operated on, as follows:

```
DATE2:     DD-MON-YY
           DD-MM-SYYYY HH24:MI:SS

DATE4:     DD-MON-YYYY
           DD-MM-SYYYY HH24:MI:SS

DATETIME: DD-MON-YYYY HH24:MI:SS
           DD-MON-YYYY HH24:MI
           DD-MM-SYYYY HH24:MI:SS
```

Note that there are multiple format masks for each type of object. For date-to-string operations, only the first-listed ("primary") mask is used. The secondary masks come into play on string-to-date operations. If the string cannot be successfully converted using the primary mask, the secondary mask(s) are tried, in the order specified above.

3. Database date format mask

Each database session within a Forms application has a single Database date format mask. It is used by the database server to convert a string to a date value, or vice versa, in the course of evaluating a query.

A default Database date format mask is established by the Oracle Server's NLS_DATE_FORMAT (or NLS_LANG or NLS_TERRITORY) initialization parameter. This can be overridden in all new database sessions for a particular client by setting the client's NLS_LANG and NLS_DATE_FORMAT environment variables.

Within a Forms application, it can be further overridden on a session-by-session basis, by issuing an ALTER SESSION.

4-6. Input, Output, and Error date format masks

The Input date format mask is used to extract a value from an item that has no format mask. That is, it is used to convert a user-entered string into a native-format date value. Rather than being a single mask, it is actually a set of masks, with different ones used in different situations.

The Output date format mask is used to display a value in an item that has no format mask, or in an LOV column. The mask is actually a pair of masks, one for DATE items and LOV columns, and one for DATETIME items. (Forms does not support DATETIME LOV columns).

The Error date format mask is used in certain error messages that can occur when an end user enters an invalid value into a date item. The mask is actually a pair of masks, one for DATE items and one for DATETIME items. For example:

FRM-50012: Date format is <date_error_format>
FRM-50025: Datetime format is <datetime_error_format>

The Input, Output, and Error date format masks are derived as follows:

Form Builder starts with the format mask derived from the current NLS environment. This may be specified explicitly by setting the environment variable NLS_DATE_FORMAT. Otherwise, NLS will compute a default format mask based on the current language. Let us call this NLS format mask "<YY_mask>".

Next, Form Builder computes a second format mask (call it "<YYYY_mask>") from <YY_mask> as follows: If <YY_mask> contains YY but not YYYY, then <YYYY_mask> is set to <YY_mask>, but with the YY replaced by YYYY. Similarly, yy will be replaced by yyyy in <YYYY_mask>. Otherwise, <YYYY_mask> is set to <YY_mask>.

Given the above, the Input format masks for DATE2, DATE4, and DATETIME objects become:

DATE2:	FXFM<YY_mask>
DATE4:	FXFM<YYYY_mask>

DATETIME: FXFM<YYYY_mask> HH24:MI:SS

FXFM<YYYY_mask> HH24:MI

(Note: FXFM indicates that whenever the format mask contains punctuation or literals, the user must enter matching punctuation and literals.)

The Output format masks become:

DATE2:   <YY_mask>

DATE4:   <YYYY_mask>

DATETIME: <YYYY_mask> HH24:MI:SS

Finally, to calculate the Error format masks, the first occurrence (if any) of RR, rr, RRRR, or rrrr in <YY_mask> and <YYYY_mask> is replaced by the corresponding number of Y's or y's.

Given the above, the Error format masks become:

DATE2:   <YY_mask>

DATE4:   <YYYY_mask>

DATETIME: <YYYY_mask> HH24:MI[:SS]

0.1.1.2 Report Builder, Releases 2.5.4.0.8, 2.5.5.2.5, 2.5.5.2.7, 2.5.7.2, 2.5.7.3

Types of Dates in Reports

Date values in Reports can be roughly categorized into three kinds:

1. Internal

   These include date values that are:

   - fetched from the data source via SQL
   - manipulated or handled with PLSQL
   - set via Value if Null for columns.

   Since these values are not exposed to the runtime user, there may be more flexibility in fixing Year 2000 issues with these date values. Localization of these Internal date format masks may be less important. This is in contrast to Output and Input date values.

2. Output

   These are exposed to a user *running* a report, but are read-only. They are formatted values displayed in Reports output to screen, files, printers, and other destination types.

3. Input

   These are exposed to a user *running* a report in the Runtime Parameter Form. They are User Parameters of the DATE type. Users may see Initial Values for date parameters and may need to type in a new date value.

Date Conversions

Reports supports a DATE datatype. Any of the Reports column types -- summary, formula, placeholder or database -- can be of the DATE datatype. There are no special conversion or data loss issues if date values are assigned to a Reports column (or PLSQL variable) of the DATE datatype.

However, there are cases where Report Builder performs date-to-string conversions and string-to-date conversions.

These conversion operations are listed in the following tables, along with the date format masks used. The individual format masks are discussed after the tables.

Table 0–4 Date-to-string operations in non-enhanced releases of Reports

Operation requiring converting a date to a string	**Format mask used**
Formatting:	
n formatting a value in a field having a format mask	The field’s format mask
n formatting a value in a field not having a format mask	Client
Runtime parameter form:	
n showing a value in a date parameter (no input mask)	Client
n showing a value in a date parameter (with input mask)	That input mask
n showing LOV values in a date parameter (no input mask)	Client
n showing LOV values in a date parameter (with input mask)	That input mask
PL/SQL V1	
n TO_CHAR(value, format_mask)	That format mask
n TO_CHAR(value) in PL/SQL	DD-MON-YY
n implicit conversion	DD-MON-YY
Database SQL (includes database SQL embedded in PL/SQL):	
n TO_CHAR(value) in a query	Database
n implicit conversion in a query	Database
PL/SQL interpreter	
n displaying the value of a local PL/SQL variable in the debugger	DD-MON-YY

Table 0–5 String-to-date operations in non-enhanced releases of Reports

Operation requiring converting a string to a date	Format mask used
Value if Null for a date value	Client
PL/SQL V1:	
■ TO_DATE(value, format_mask)	That format mask
■ TO_DATE(value) in PL/SQL	DD-MON-YY
■ implicit conversion in PL/SQL	DD-MON-YY
Database SQL (includes database SQL embedded in PL/SQL):	
■ TO_DATE(value) in a query	Database
■ implicit conversion in a query	Database
PL/SQL interpreter:	
■ setting the value of a local PL/SQL variable in the debugger	DD-MON-YY

Types of date format masks

The non-enhanced releases of Report Builder use the following date format masks:

1. User-created date format masks for individual items
2. Database date format mask
3. Client date format mask

Each mask is discussed below.

There also is an internal, intermediate-level date format mask value, referred to here as the **NLS date format mask**. This mask value is derived from NLS_xxx environment variables, and it is then used to establish the Database and Client date format masks (as explained below). In adddition, this NLS date format mask value is used for the following:

- DATE field default format mask
- DATE column’s Value If Null
- DATE User Parameter default input mask

1. User-created date format masks for individual items

These are set at design time, typically through the property palette. For fields in the layout, they can also be set at runtime through the SRW.SET_FORMAT_MASK property.

To the extent possible, individual objects date format masks should *not* be created. Hard-coding individual masks makes internationalization difficult.

2. Database date format mask

Each database session within a Reports application has a single Database date format mask. This mask is used by the database server to convert a string to a date value, or vice versa, in the course of evaluating a query.

The Database date format mask is established by the server's NLS_DATE_FORMAT environment variable, if it has been set. Otherwise, it is derived from the NLS_LANG and NLS_TERRITORY environment variables. (In the non-date-format-enhanced releases of Report Builder, this default value derived from NLS_LANG and NLS_TERRITORY is based on a 2-digit year format.)

Note: when setting NLS_DATE_FORMAT, it is important to also explicitly set NLS_LANG. Otherwise, NLS_DATE_FORMAT will be ignored for the purposes of defining the Database date format mask.

Within a Reports application, the NLS_xxx-derived Database date format mask can be overridden on a session-by-session basis by issuing an ALTER SESSION.

3. Client date format mask

A value for NLS_DATE_FORMAT (and/or NLS_LANG and NLS_TERRITORY) can also be specified on the client side. (This is independent of these settings on the server side.) This setting establishes the value of the Client mask. The semantics of setting these environment variables are the same as for the Database date format mask on the server

0.1.1.3 Graphics Builder, Release 1.6

Graphics functions as a client, and date handling is done primarily in that context. For the most part, Graphics follows the PL/SQL conventions and formats for date conversions. Date considerations arise in the following areas.

1. Data Table internal display

   The Date Table is part of the query dialog display. The table displays the first 1500 rows fetched from the query.

   The user can control how dates are displayed that come from SQL queries. This is done through the format on the query property dialog. Internally, dates are handled as type DATE; therefore, the year format is just to make them more readable for the developer.

   For file queries (prn-type files, and so forth), the designer can indicate which columns are to be interpreted as dates.

2. **Discrete Axis tick labels**

   Columns plotted on a discrete axis as treated as character strings. Therefore, the values as they appear in the Data Table are used as the tick label.

3. **Date Axis tick labels**

   The labels are set via the Date Axis dialog. This affects only the labels, not the values.

4. **Date Axis min/max/step values**

   It is not necessary to set date values here unless the user wants to override the defaults. Overriding can be done via the Date Axis property dialog. The user is limited to the DD-MON-YY format.

5. **Date chart element labels**

   The menu option format/date format allows the display of a label for each chart element plotted that reflects the data value (which may be a date). the appearance of the date information is only for readability, since internally no data is lost.

6. **Date parameter**

   Date parameters can be created, and their initial value established in the parameter dialog. The designer can also use the Graphics built-ins og_get_date_param and og_set_param to set this value programmatically. In all cases, the date format must agree with the display property date format.

7. **Date property of object**

Objects in Graphics can have dynamically-created user-defined properties attached to them. These properties can be of type DATE (as well as CHAR and NUMBER). However, for date properties only the YY value is stored.

8. Text objects that are dates

Text objects that are display only have a default expandable bounding box characteristic. Input text objects have a default fixed bounding box characteristic.

These behaviors can be modified via the menu option Format/drawing option/text drawing option.

9. PL/SQL

Graphics allows users to write client-side PL/SQL program units. In such units, the handling of dates follows PL/SQL rules.

0.1.2 Dates in Oracle Developer -- Enhanced Releases

This section covers date handling in Releases 1.3.3, 1.6.1, 2.1, and 6.0 of Oracle Developer. Forms, Reports, and Graphics are each discussed in their own sections.

Additional Information

If you need a refresher on the specific mask elements in a date format mask, see the section “Date Format Models” in your *Oracle Server SQL Reference* manual.

0.1.2.1 Form Builder, Releases 4.5.7.18.0, 4.5.10, 5.0.6, and 6.0.x

The handling of dates in these enhanced releases of Form Builder depends on several things: on the type of date object involved, on the conversion operation being performed, and especially on which of the many format masks is being used.

Types of date objects

In Form Builder, a date object is considered to be either a DATE2 object, a DATE4 object, or a DATETIME object. The following table shows this categorization. In this paper, these objects are referred to collectively as date objects.

Table 0–6   Types of date objects in enhanced releases

Date object	Type
Item of datatype DATETIME	DATETIME
Item of datatype DATE:	
▪ having a format mask that contains yyyy, YYYY, rrrr, or RRRR	DATE4
▪ having a format mask that does not contain yyyy, YYYY, rrrr, or RRRR	DATE2
▪ not having a format mask, and of length (Maximum Length) of 9 or less	DATE2
▪ not having a format mask, and of length (Maximum Length) of 10 or more	DATE4
Parameter (as in :PARAMETER.myparam) of datatype DATE. (Note that there are no DATETIME parameters, and that a parameter's Maximum Length property applies only to CHAR parameters.)	DATE2
LOV column of datatype DATE. (Note that there are no DATETIME LOV columns.)	DATE2
Internal value of system variable CURRENT_DATETIME and EFFECTIVE_DATE	DATETIME

Types of conversion operations

Form Builder performs conversion operations in several circumstances:

- when converting a date to a string
- when converting a string to a date
- when converting a user-entered date for display in an error message.

The two tables below list the various conversion operations. They also show which format masks are used. The format masks themselves are explained after the tables.

Table 0–7 Date-to-string operations in enhanced releases of Forms

Operation requiring converting a date to a string	Format mask used
Formatting:	
- Displaying a value in an item that has a format mask	The item’s format mask
- Displaying a value in an item that has no format mask	Output
- Displaying a value in a date LOV column	Output
Internal Forms operations:	
- Getting :SYSTEM.CURRENT_DATETIME	Builtin
- Getting :SYSTEM.EFFECTIVE_DATE	Builtin
- Setting a CHAR return item from a date LOV column	Builtin (but see note 1)
- Setting a CHAR item's initial value from $$DATE$$, $$DATETIME$$, $$DBDATE$$, or $$DBDATETIME$$	Builtin (but see note 1)
- GET_ITEM_PROPERTY(item, DATABASE_VALUE)	Builtin (but see note 1)
- GET_ITEM_PROPERTY(item, ITEM_DEFAULT_VALUE)	Builtin (but see note 1)
- GET_ITEM_PROPERTY(item, RANGE_HIGH/LOW)	Builtin (but see note 1)
- GET_LIST_ELEMENT_VALUE(item, value)	Builtin
- GET_PARAMETER_ATTR(list, key, paramtype, value) for a date parameter defined at design time	Builtin
- NAME_IN(variable)	Builtin

Table 0–7 Date-to-string operations in enhanced releases of Forms

Operation requiring converting a date to a string	**Format mask used**
PL/SQL V1:	
n TO_CHAR(value, format_mask) in PL/SQL	That format_mask (see note 2)
n TO_CHAR(value) in PL/SQL	PLSQL
n Implicit conversion in PL/SQL	PLSQL
Database SQL (includes database SQL embedded in PL/SQL):	
n TO_CHAR (value, format mask) in database SQL	That format mask
n TO_CHAR (value) in database SQL	Database
n Implicit conversion in database SQL	Database
n Storing a date value returned by database SQL into a Forms CHAR variable	Database
n Converting a Forms date variable when it needs to be inserted into or compared against a CHAR column	Database
PL/SQL debugger:	
n Displaying the value of an item or a parameter in the debugger	Builtin
n Displaying the value of a local PL/SQL variable in the debugger	PLSQL

(Notes follows next table.)

-

Table 0–8 String-to-date operations in enhanced releases of Forms

Operation requiring converting a string to a date	Format mask used
Formatting:	
▪ Extracting a value from an item that has a format mask	The item's format mask
▪ Extracting a value from an item that has no format mask	Input
Internal Forms operations:	
▪ Setting :SYSTEM.EFFECTIVE_DATE	Builtin
▪ Setting a DATE or DATETIME return item from a CHAR LOV column	Builtin (but see note 1)
▪ Setting a DATE or DATETIME item's initial value from a CHAR variable (using colon notation)	Builtin (but see note 1)
▪ ADD_LIST_ELEMENT(item, index, label, value)	Builtin
▪ SET_PARAMETER_ATTR(list, key, paramtype, value) for a date parameter defined at design time	Builtin
▪ COPY(value, variable)	Builtin
▪ DEFAULT_VALUE(value, variable)	Builtin
PL/SQL V1:	
▪ TO_DATE(value, format_mask) in PL/SQL	That format mask (see note 2)
▪ TO_DATE(value) in PL/SQL	PLSQL
▪ Implicit conversion in PL/SQL	PLSQL
Database SQL (includes database SQL embedded in PL/SQL):	
▪ TO_DATE(value, format_mask) in database SQL	That format mask
▪ TO_DATE(value) in database SQL	Database
▪ Implicit conversion in database SQL	Database
▪ Storing a CHAR value returned by database SQL into a Forms DATE variable	Database
▪ Converting a Forms CHAR variable when it needs to be inserted into or compared against a date column	Database
PL/SQL debugger:	
▪ Setting the value of an item or a parameter in the debugger	Builtin
▪ Setting the value of a local PL/SQL variable in the debugger	PLSQL

Table notes:

1. If the effective date format compatibility mode for the form is set to 4.5, then these operations use the Output (for date-to-string) or Input (for string-to-date) format mask instead of the Builtin format mask.

   In Oracle Developer Releases 1.3.3 and 1.6.1, the effective date format compatibility mode is simply the current value of the application's DATE_FORMAT_COMPATIBILITY_MODE property. This property can be programmatically set to 4.5 or 5.0. It defaults to 4.5.

   In Oracle Developer Release 2.1 and above, the effective date format compatibility mode is determined as in Releases 1.3.3 and 1.6.1, except that when the current form's RUNTIME_COMPATIBILITY_MODE is 5.0, the effective date format compatibility mode is always 5.0, regardless of the current value of the application's DATE_FORMAT_COMPATIBILITY_MODE property. The RUNTIME_COMPATIBILITY_MODE property is settable at design time in Oracle Developer Release 2.0 and above. It defaults to 5.0 for new forms and to 4.5 for forms upgraded from Forms 4.5.

   It is strongly recommended that new applications set the DATE_FORMAT_COMPATIBILITY_MODE property to 5.0.

2. In contrast to earlier, non-date-format-enhanced releases of Form Builder, a format mask specified in TO_CHAR and TO_DATE constructs in PL/SQL or database SQL may contain the RR or RRRR format mask element.

Types of date format masks

The enhanced releases of Form Builder use the following date format masks:

1. User-created date format masks for individual items
2. PLSQL date format mask
3. Builtin date format mask
4. Database date format mask
5. Input date format mask
6. Output date format mask
7. Error date format mask

Each mask is discussed below.

1. Individual items’ date format masks

These are set by the programmer or user at design time or runtime, through the Format_Mask property.

Note that there is no requirement that an individual item have its own individual mask. Indeed, where possible this should be avoided, since the presence of such masks makes internationalization difficult. (Items without individual masks will be processed by Form Builder using its general-purpose Input and Output masks.)

2. PLSQL data format mask

The PLSQL date format mask is a single format mask that is global to the entire Forms application.

You set and inspect this mask in the PLSQL_DATE_FORMAT property. The property is accessible via the built-in subprograms SET_APPLICATION_PROPERTY and GET_APPLICATION_PROPERTY.

It is strongly recommended that new applications set this property to a format mask containing full century and time information. It is further recommended that this format mask be the same as the one specified for the BUILTIN_DATE_FORMAT property (see below). For example, the PRE-LOGON trigger in the application’s initial form could contain:

```
SET_APPLICATION_PROPERTY(PLSQL_DATE_FORMAT,
                                        'YYYY/MM/DD HH24:MI:SS');
SET_APPLICATION_PROPERTY(BUILTIN_DATE_FORMAT,
                                        'YYYY/MM/DD HH24:MI:SS');
```

For compatibility with prior releases, the application’s PLSQL_DATE_FORMAT property defaults to a value of "DD-MON-YY". But be forewarned that if you use this format mask, it will typically be more difficult to code new program units so as to avoid losing century and time information. It may also be more difficult to avoid datatype conversion errors.

Note that in Oracle products other than forms, PL/SQL version 2 does not necessarily use a default date format mask of "DD-MON-YY". Instead, it typically uses a format mask derived from the current NLS environment. If, for some reason, you wish your Forms application to exhibit the same behavior, you can use the new application property USER_NLS_DATE_FORMAT to get the current NLS date format mask, and then assign it to the application’s PLSQL_DATE_FORMAT property. For example, the PRE-FORM trigger in the application’s initial form could contain:

```
SET_APPLICATION_PROPERTY(PLSQL_DATE_FORMAT,
GET_APPLICATION_PROPERTY(USER_NLS_DATE_FORMAT));
```

The USER_NLS_DATE_FORMAT property is read-only; that is, you may not specify it in SET_APPLICATION_PROPERTY.

3. Builtin date format mask

The Builtin date format mask is used for internal operations within Forms when converting a date value to or from a string that is not potentially visible to the end user.

This mask can be set and inspected via the built-in subprograms SET_APPLICATION_PROPERTY and GET_APPLICATION_PROPERTY, using the BUILTIN_DATE_FORMAT property. Once set, the Builtin date format mask is a single format mask which is global to the entire Forms application.

It is strongly recommended that new applications set this property to a format mask containing full century and time information. It is further recommended that this format mask be the same as the one specified for the PLSQL_DATE_FORMAT property (see above). For example, the PRE-LOGON trigger in the application's initial form could contain:

```
SET_APPLICATION_PROPERTY(PLSQL_DATE_FORMAT,
                         'YYYY/MM/DD HH24:MI:SS');

SET_APPLICATION_PROPERTY(BUILTIN_DATE_FORMAT,
                         'YYYY/MM/DD HH24:MI:SS');
```

There are two special values which may be specified for the BUILTIN_DATE_FORMAT application property, that are not treated in the normal way:

YY: Indicates that date format masks for internal date-string conversions will be chosen based on the type of date object, as follows:

```
DATE2:     DD-MON-YY
           DD-MM-SYYYY HH24:MI:SS

DATE4:     DD-MON-YYYY
           DD-MM-SYYYY HH24:MI:SS

DATETIME: DD-MON-YYYY HH24:MI:SS
           DD-MON-YYYY HH24:MI
           DD-MM-SYYYY HH24:MI:SS
```

Note that there are multiple format masks for each type of object. For date-to-string operations, only the first-listed ("primary") mask is used. The secondary format masks come into play on string-to-date operations. If the string cannot be successfully converted using the primary mask, the secondary mask(s) are tried, in the order specified above.

YY is the default (initial) value of the BUILTIN_DATE_FORMAT application property. It provides compatibility with prior releases.

RR: Indicates that date format masks for internal date-string conversions will be chosen based on the type of date object, as follows:

DATE2:	DD-MON-RR
	DD-MM-SYYYY HH24:MI:SS
DATE4:	DD-MON-RRRR
	DD-MM-SYYYY HH24:MI:SS
DATETIME:	DD-MON-RRRR HH24:MI:SS
	DD-MON-RRRR HH24:MI
	DD-MM-SYYYY HH24:MI:SS

The 'RR' masks are identical to the 'YY' masks, except that DD-MON-YY is replaced by DD-MON-RR, and DD-MON-YYYY is replaced by DD-MON-RRRR.

Be forewarned that if you don't explicitly set the BUILTIN_DATE_FORMAT property (to something other than 'YY'), then (as for the PLSQL_DATE_FORMAT property), it will typically be more difficult to code new program units so as to avoid losing century and time information, and it may also be more difficult to avoid datatype conversion errors.

4. Database date format mask

Each database session within a Forms application has a single Database date format mask. It is used by the database server to convert a string to a date value, or vice versa, in the course of evaluating a query.

A default Database date format mask is established by the Oracle Server's NLS_DATE_FORMAT (or NLS_LANG or NLS_TERRITORY) initialization parameter. This can be overridden in all new database sessions for a particular client, by setting the client's NLS_LANG and NLS_DATE_FORMAT environment variables.

Within a Forms application, it can be further overridden on a session-by-session basis, by issuing an ALTER SESSION. It may simplify an application's logic if it is set to the same format mask as the application's PLSQL_DATE_FORMAT and BUILTIN_DATE_FORMAT properties. For example, the PRE-LOGON trigger in the application's initial form could contain:

```
SET_APPLICATION_PROPERTY(PLSQL_DATE_FORMAT,
                                        'YYYY/MM/DD HH24:MI:SS');
```

```
SET_APPLICATION_PROPERTY(BUILTIN_DATE_FORMAT,
                                        'YYYY/MM/DD HH24:MI:SS');
```

and the POST-LOGON trigger could contain:

```
FORMS_DDL('ALTER SESSION SET NLS_DATE_FORMAT =
                                    ''YYYY/MM/DD HH24:MI:SS''');
```

Note that if you do an OPEN_FORM with the SESSION option specified, then the opened form will have a new database session, so you might want to alter its session as well.

5. Input date format mask

The Input date format mask is used at runtime to extract a value from an item that has no individual date format mask. That is, it is used to convert a user-entered string into a native-format date value. This mask is also used by Forms at design time, to convert a date entered in the property palette into native format.

The mask is actually a set of masks, with different ones used in different situations.

The system administrator or end user may set the environment variables FORMSnn_USER_DATE_FORMAT and FORMSnn_USER_DATETIME_FORMAT (where nn = 45, 50, or 60, depending on the release of Forms) to specify the format masks used to extract values from items of type DATE and DATETIME, respectively. In each case, multiple format masks may be specified, separated by vertical bars. When converting a user-entered string, the format masks are used in the order specified, until a conversion succeeds or until the format masks are exhausted.

For example, you might set these two environment variables as follows:

```
 FORMSnn_USER_DATE_FORMAT:
             FXFMDD-MM-RRRR
 FORMSnn_USER_DATETIME_FORMAT:
             FXFMDD-MM-RRRR  HH24:MI:SS|FXFMDD-MM-RRRR HH24:MI
```

This would force the end user to enter values into DATE items (with no format mask) in the format exemplified by 31-6-97. (The RRRR enables years between 1950 and 2049 to be entered with the century omitted). But for DATETIME items, the end user could enter values either in the format exemplified by 31-6-97 13:45:30 or else in the format exemplified by 31-6-97 13:45 (which would be interpreted as 31-6-97 13:45:00).

If either of these two environment variables is not defined, then Forms will construct default format masks as documented below in the section "default input, output, and error date format masks".

The effective input format mask(s) for DATE4 and DATETIME items can be obtained from the application properties USER_DATE_FORMAT and USER_DATETIME_FORMAT. These properties are read-only. "Effective" means the appropriate default value will be returned in cases where either of the environment variables FORMSnn_USER_DATE_FORMAT or FORMSnn_USER_DATETIME_FORMAT are not defined. In the case where the effective value contains more than one input mask, the value of the application property is obtained by concatenating them, separated by vertical bars.

Note that the effective input format mask for DATE2 items is the same as for DATE4 items if FORMSnn_USER_DATE_FORMAT is defined.

Installations may wish to establish a standard that multiple format masks are not allowed in the values specified for the FORMSnn_USER_DATE_FORMAT and FORMSnn_USER_DATETIME_FORMAT environment variables (that is, the values must not contain vertical bars). This will simplify the logic of applications that inspect the USER_DATE_FORMAT and USER_DATETIME_FORMAT application properties. Such a standard is more likely to make sense in Oracle Developer Release 1.6.1 than in Releases 1.3.3, 2.1, and 6.0, because Release 1.6.1 supports "flexible date formatting" on string-to-date operations when the format mask does not contain FX. "Flexible date formatting" means that if an initial conversion attempt fails, then the conversion will be attempted using various modifications to the format mask (e.g., substituting MON for MM or vice versa, or omitting punctuation and/or trailing time format mask elements such as SS, MI, and HH). See the Release Notes for specific Form Builder releases for details.

6. Output date format mask

The "output" date format mask is used at runtime to display a value in an LOV column or in an item that has no format mask. The mask is also used by Forms at design time, to display dates in the property palette.

This mask is actually a pair of format masks, one for DATE items and LOV columns, and one for DATETIME items. (Form Builder does not support DATETIME LOV columns).

The system administrator or end user may set the environment variables
FORMSnn_OUTPUT_DATE_FORMAT and
FORMSnn_OUTPUT_DATETIME_FORMAT
to specify these format masks explicitly.

If FORMSnn_OUTPUT_DATE_FORMAT is not defined, but FORMSnn_USER_DATE_FORMAT *is* defined, then the Output date format mask for DATE items is derived from FORMSnn_USER_DATE_FORMAT by taking the first format mask (the value up to the first vertical bar) and stripping out all occurrences of FX and FM.

For example, if FORMSnn_OUTPUT_DATE_FORMAT is not defined and FORMSnn_USER_DATE_FORMAT is defined to be FXFMDD-MM-RRRR|FXFM-MON-RRRR, then the Output date format mask for DATE items is DD-MM-RRRR.

If neither FORMSnn_OUTPUT_DATE_FORMAT nor FORMSnn_USER_DATE_FORMAT is defined, then Forms will construct default format masks as documented below in the section *Default input, output, and error date format masks* below.

There are similar rules for DATETIME items.

The effective Output format mask(s) for DATE4 and DATETIME items can be obtained from the application properties OUTPUT_DATE_FORMAT and OUTPUT_DATETIME_FORMAT. These properties are read-only. "Effective" means the appropriate default value will be returned in cases where either of the environment variables FORMSnn_OUTPUT_DATE_FORMAT or FORMSnn_OUTPUT_DATETIME_FORMAT are not defined.

Note that the effective Output format mask for DATE2 items is the same as for DATE4 items if either FORMSnn_OUTPUT_DATE_FORMAT or FORMSnn_USER_DATE_FORMAT is defined.

7. Error date format mask

The Error date format mask is used in generating certain error messages that are issued when an end user enters an invalid value into a date item. For example:

FRM-50012: Date must be entered in a format like <date_error_format>

FRM-50025: Date/time must be entered in a format like <datetime_error_format>

The exact error messages may differ from release to release.

This mask is actually a pair of masks, one for DATE items and one for DATETIME items.

The system administrator or end user may set the environment variables FORMSnn_ERROR_DATE_FORMAT and FORMSnn_ERROR_DATETIME_FORMAT to specify these format masks explicitly.

If FORMSnn_ERROR_DATE_FORMAT is not defined, but FORMSnn_USER_DATE_FORMAT *is* defined, then the Error date format mask for DATE items is simply taken to be the entire value FORMSnn_USER_DATE_FORMAT, including vertical bars.

If neither FORMSnn_ERROR_DATE_FORMAT nor FORMSnn_USER_DATE_FORMAT is defined, then Forms will construct default format masks as documented below in the section *Default input, output, and error date format masks* below.

There are similar rules for DATETIME items.

Default Input, Output, and Error date format masks

If the environment variables FORMSnn_USER_DATE_FORMAT and FORMSnn_USER_DATETIME_FORMAT are not both set, then the rules above do not specify all of the Input, Output, and Error format masks for DATE items and DATETIME items. These remaining unspecified format masks are determined using the rules below.

NOTE: It is recommended that the environment variables FORMSnn_USER_DATE_FORMAT and FORMSnn_USER_DATETIME_FORMAT both be set. In that case, the complex rules below will never come into play. The behavior described below is intended primarily to provide compatibility with non-date-format-enhanced releases, although the date-format-enhanced releases do contain some minor refinements.

Forms starts with the format mask derived from the current NLS environment. This may be specified explicitly by setting the environment variable NLS_DATE_FORMAT. Otherwise, NLS will compute a default format mask based on the current language. Let us call this NLS format mask "<YY_mask>".

Next, Forms computes a second format mask (call it "<YYYY_mask>") from <YY_mask> as follows: If <YY_mask> contains YY but not YYYY, then <YYYY_mask> is set to <YY_mask>, but with the YY replaced by YYYY. Similarly, yy, RR, or rr will be replaced by yyyy, RRRR, or rrrr in <YYYY_mask>. Otherwise, <YYYY_mask> is set to <YY_mask>. (Note: this behavior differs from the non-enhanced releases of Forms, in which RR and rr were not changed to RRRR and rrrr.)

Given the above, the remaining unspecified Input format masks for DATE2, DATE4, and DATETIME objects become:

DATE2:	FXFM<YY_mask>
DATE4:	FXFM<YYYY_mask>
DATETIME:	FXFM<YYYY_mask> HH24:MI:SS FXFM<YYYY_mask> HH24:MI

The leading FXFM is omitted if the format mask already contains an FX or FM. (Note: FXFM indicates that whenever the format mask contains punctuation or literals, the user must enter matching punctuation and literals.)

Next, any occurrences of FX and FM are removed from <YY_mask> and <YYYY_mask>. (Note: this was not done in the non-enhanced releases of Forms.)

Given the above, the remaining unspecified Output format masks become:

DATE2: <YY_mask>

DATE4: <YYYY_mask>

DATETIME: <YYYY_mask> HH24:MI:SS

Finally, for the remaining unspecified Error format masks, the first occurrence (if any) of RR, rr, RRRR, or rrrr in <YY_mask> and <YYYY_mask> is replaced by the corresponding number of Y's or y's.

Given the above, the remaining unspecified Error format masks become:

DATE2: <YY_mask>

DATE4: <YYYY_mask>

DATETIME: <YYYY_mask> HH24:MI[:SS]

0.1.2.1.1 Form Builder Release 5.0.5

As noted earlier, this release of Form Builder has most of the date-handling enhancements explained above -- but not all. In Release 5.0.5:

- These application properties are *not* supported:
  - USER_DATE_FORMAT
  - USER_DATETIME_FORMAT
  - OUTPUT_DATE_FORMAT
  - OUTPUT_DATETIME_FORMAT
  - DATE_FORMAT_COMPATIBILITY_MODE
- The BUILTIN_DATE_FORMAT application property cannot be set to YY or RR. Its initial value is the empty string (which indicates the same behavior as does YY in fully-enhanced releases).
- A change to the PLSQL_DATE_FORMAT application property does not affect currently-executing or suspended triggers. (This is bug 607256.)

0.1.2.2 Report Builder, Releases 2.5.5.20, 2.5.7.4.3, 2.5.7.5, and 3.0.5

These are the date-format-enhanced releases of Report Builder.

Types of Dates in Reports

Date values in Reports can be roughly categorized into three kinds:

1. Internal

   These include date values that are:

   - fetched from the data source via SQL
   - manipulated or handled with PL/SQL
   - set via Value if Null for columns.

   Since these values are not exposed to the runtime user, there may be more flexibility in fixing Year 2000 issues with these date values. Localization of these Internal date format masks may be less important. This is in contrast to Output and Input date values.

2. Output

   These are exposed to a user *running* a report, but are read-only. They are formatted values displayed in Reports output to screen, files, printers or other destination types.

3. Input

   These are exposed to a user *running* a report in the Runtime Parameter Form. They are User Parameters of the DATE type. Users may see Initial Values for date parameters and may need to type in a new date value.

Date format enhancements

The date-format-enhanced releases of Report Builder have the following features:

1. RR and RRRR elements are supported in TO_DATE and TO_CHAR calls in PL/SQL and in SQL database queries.
2. The Value If Null for DATE columns can now be set through a new command argument, NVL_DATE_FORMAT. The value for these columns is determined using the following precedence order:

   - The NVL_DATE_FORMAT value is used, if one was specified.

- The NLS date format value is used, if one was specified.
- DD-MON-YY is used if neither of the above was specified.

Note that DATE User Parameters are *not* governed by the NVL_DATE_FORMAT setting. These parameters support an input mask setting that can be used to specify the format of a date in Initial Value.

3. The mask for implicit date-to-string and string-to-date conversions can now be set through a new command argument, PLSQL_DATE_FORMAT. This date format mask's value is determined using the following precedence order:

- The PLSQL_DATE_FORMAT value is used, if one was specified.
- The NLS date format value is used, if one was specified. (This is true only in Oracle Developer Release 2.0 using PL/SQL V2 and above.)
- DD-MON-YY is used if neither of the above was specified.

Note that NLS date format is not used in PL/SQL V1 for implicit conversions.

By setting these command line keywords, users can control date-to-string conversions more precisely. Note that these settings affect *all* relevant date conversions during the running of the report. (See the table below.)

For compatibility, these new command line keywords will be supported in future Report Builder releases.

In addition to these command line keywords, PL/SQL V1 has been enhanced to support the RR and RRRR format masks.

Date conversions

Reports supports a DATE datatype. Any of the Reports column types -- summary, formula, placeholder or database -- can be of the DATE datatype. There are no special conversion or data loss issues if date values are assigned to a Reports column (or PLSQL variable) of the DATE datatype.

However, there are cases where Report Builder performs date-to-string conversions and string-to-date conversions occur.

These conversion operations are listed in the following tables, along with the date format masks used. The individual format masks are discussed after the tables.

Table 0–9 Date-to-string operations in enhanced releases of Reports

Operation requiring converting a date to a string	**Format mask used**
Formatting:	
▪ formatting a value in a field having a format mask	The field’s format mask
▪ formatting a value in a field not having a format mask	Client
Runtime parameter form:	
▪ showing a value in a date parameter (no input mask)	Client
▪ showing a value in a date parameter (with input mask)	The input mask
▪ showing LOV values in a date parameter (no input mask)	Client
▪ showing LOV values in a date parameter (with input mask)	The input mask
PL/SQL V1	
▪ TO_CHAR(value, format_mask)	That format mask
▪ TO_CHAR(value, ‘RR_format_mask’)	RR_format_mask *
▪ TO_CHAR(value) in PL/SQL, with PLSQL_DATE_FORMAT unspecified	DD-MON-YY
▪ TO_CHAR(value) in PL/SQL, with PLSQL_DATE_FORMAT specified	That format mask *
▪ implicit conversion in PL/SQL, with PLSQL_DATE_FORMAT unspecified	DD-MON-YY
▪ implicit conversion in PL/SQL, with PLSQL_DATE_FORMAT specified	That format mask *
Database SQL (includes database SQL embedded in PL/SQL):	
▪ TO_CHAR(value) in a query	Database
▪ implicit conversion in a query	Database
PL/SQL interpreter	
▪ displaying the value of a local PL/SQL variable in the Debugger	DD-MON-YY

(Notes follow next table.)

Table 0–10 String-to-date operations in enhanced releases of Reports

Operation requiring converting a string to a date	Format mask used
Value if Null for a date value:	
n with NVL_DATE_FORMAT unspecified	Client
n with NVL_DATE_FORMAT specified	That format mask *
PL/SQL V1:	
n TO_DATE(value, format_mask)	That format mask
n TO_DATE(value) in PL/SQL	DD-MON-YY
n implicit conversion in PL/SQL	DD-MON-YY
Database SQL (includes database SQL embedded in PL/SQL):	
n TO_DATE(value) in a query	Database
n implicit conversion in a query	Database
PL/SQL interpreter:	
n setting the value of a local PL/SQL variable in the Debugger	DD-MON-YY

Note: in the table, the items flagged with an asterisk (*) have new/changed behavior from earlier, non-date-format-enhanced releases of Report Builder.

Types of date format masks

The enhanced releases of Report Builder use the following date format masks:

1. User-created date format masks for individual items
2. Database date format mask
3. Client date format mask

Each mask is discussed below.

There also is an internal, intermediate-level date format mask value, referred to here as the **NLS date format mask**. This mask value is derived from NLS_xxx environment variables, and it is then used to establish the Database and Client date format masks (as explained below). In adddition, this NLS date format mask value is used for the following (although it is possible to override these values in various ways):

- PL/SQL implicit conversion date format mask (for Oracle Developer Release 2.0 and above only)
- DATE field default format mask
- DATE column’s Value If Null
- DATE User Parameter default input mask

1. User-created date format masks

These are set at design time, typically through the property palette. For fields in the layout, they can also be set at runtime through the SRW.SET_FORMAT_MASK property.

Where possible, individual objects date format masks should not be set. Hard-coding individual masks makes internationalization difficult.

2. Database date format mask

Each database session within a Reports application has a single Database date format mask. This mask is used by the database server to convert a string to a date value, or vice versa, in the course of evaluating a query.

The Database date format mask is established by the server's NLS_DATE_FORMAT environment variable, if it has been set. Otherside, it is derived from the NLS_LANG and NLS_TERRITORY environment variables.

Note: when setting NLS_DATE_FORMAT, it is important to also explicitly set NLS_LANG. Otherwise, NLS_DATE_FORMAT will be ignored for the purposes of defining the Database date format mask.

Within a Reports application, the Database date format mask can be further overridden on a session-by-session basis by issuing an ALTER SESSION.

3. Client date format mask

A value for NLS_DATE_FORMAT (and/or NLS_LANG and NLS_TERRITORY) can also be specified on the client side (this is independent of these settings on the server side). This setting establishes the value of the Client mask. The semantics of setting these environment variables are the same as for the database server

0.1.2.3 Graphics Builder, Releases 2.5.x and 3.0.x

In contrast to Form Builder and Report Builder, the treatment of dates in Graphics Builder has not changed in its recent releases. Therefore, the descriptions given above for Graphics in the non-enhanced section of this paper also apply here.

However, there are additional considerations in the area of user-coded PL/SQL program units. Later releases of Graphics use later versions of PL/SQL that support the RR and RRRR date formats.

0.1.3 Year 2000 Suggestions for Developer Applications -- Non-Enhanced Releases

Upgrade Recommendation

If you are currently using one of these non-date-enhanced releases of Oracle Developer, it is strongly recommended that you upgrade to a later release that has the enhancements. Using the later releases simplifies the task of dealing with Year 2000 problems in particular. It has the added advantage of reducing date-handling complexities in general.

If upgrading to an enhanced release is not possible, use the suggestions in this section to help you move towards Year 2000 compliance in your Oracle Developer applications.

Suggestions are offered separately for Form Builder, Report Builder, and Graphics Builder.

To get maximum value from the following material, you may wish to first read the earlier general section on how dates are handled in the non-enhanced releases of Oracle Developer.

0.1.3.1 Suggestions for Forms Applications

These suggestions are for the non-date-enhanced releases of Form Builder: Releases 4.5.6, 4.5.7.0 to 4.5.7.16.2, 4.5.8, and 4.5.9

The suggestions are divided into those for creating new applications and those for modifying existing applications. These are preceded by suggestions for setting environment variables for your forms builder and compiler/generator.

(For definitions of the DATE2, DATE4, and DATETIME date objects, see the table on page -7. For the various date format masks used by Form Builder, see the descriptions starting on page -11.)

0.1.3.1.1 Suggestions for the Build and Compile Environment

Whether you are creating a new application or working on an existing one, you should set the DATE variable for the environment in which Form Builder will operate during its design and compile stages.

Specifically, the NLS_DATE_FORMAT environment variable should be set to a format mask containing YYYY. This ensures that centuries in dates (specified as the values of properties) will be preserved when you open and save an fmb file at design-time, and when you compile and generate an fmx file. (Note that this guideline applies in the runtime environment for the generated application, as well, as explained below.)

0.1.3.1.2 About Customized Forms

Not all of the suggestions in this white paper apply to forms that have been customized and are part of an Oracle Applications product. For more information, consult the Oracle Applications documentation or an Oracle Applications support representative.

0.1.3.1.3 Suggestions for New Forms Applications

If you are designing and coding a new application using a non-date-enhanced release of Form Builder, do the following:

1. Never use DATE2 objects.
2. In your PL/SQL coding:

- Never code a statement that could cause an implicit conversion from a string to date, or vice versa. Instead, include an explicit TO_DATE or TO_CHAR construct in the statement.
- In all TO_DATE and TO_CHAR constructs, always specify an explicit format mask containing YYYY.

3. Never use the RR or RRRR format mask element in the PL/SQL TO_DATE or TO_CHAR constructs. (However, these are permissible in database SQL).
4. Never use the YY format mask element.
5. If your application uses any positive or negative "infinity" dates (artificial dates that are meant to be higher or lower than any normal date your application might manipulate), make sure that they will still work properly in the 21st century. (So, for example, Dec 31, 1999 is not an appropriate positive infinity date.) The recommended positive infinity date is the highest date representable by the Oracle DATE datatype in all releases of Oracle Developer; namely, Dec 31, 4712. The recommended negative infinity date is Jan 1, 100 A.D. You can code these (for example) as:

   TO_DATE('4712/12/31','YYYY/MM/DD') and
   TO_DATE('0100/01/01','YYYY/MM/DD').

   Any positive or negative infinity dates stored in your database should also conform to this guideline.
6. In the runtime environment that your application will execute in, set the NLS_DATE_FORMAT environment variable to a value containing YYYY.

0.1.3.1.4 Suggestions for Existing Forms Applications:

1. All DATE2 items must be converted to DATE4 items. Any logic which produces a string from such an item (for example, using NAME_IN) or which sets the item's value from a string (for example, using COPY) must be modified to accommodate the fact that the string will (or must) contain a 4-digit year rather than a 2-digit year. Local PL/SQL variables that are used to hold such strings may need to be enlarged.

If the resulting DATE4 item has a format mask, the item’s Maximum Length property must be set to a value large enough to accommodate the maximum length string that can be produced by the format mask element.

When increasing an item's Maximum Length property, you may wish to increase the item's Width property so that the entire value is visible. Of course, this may require that adjacent items be moved.

Note that in Oracle Developer Release 1.2, it is theoretically possible to create a DATE4 item with a format mask containing the YY format mask element. However, do not do this for text items. (Although this is permissible for display items, it will seldom make sense. If Maximum Length is 11 or more, it is preferable to use YYYY in the item’s format mask.)

2. In the application's PL/SQL code, all TO_DATE and TO_CHAR constructs that do not specify a format mask (or that do specify a format mask containing YY) must be altered to have an explicit format mask containing YYYY. All application logic that manipulates these strings must be modified to accommodate the new format.

Hard-coded dates often violate this guideline. For example, consider TO_DATE(‘01-jan-20’). This should be converted to TO_DATE(‘1920/01/01’, ‘YYYY/MM/DD’).

3. An explicit format mask containing YYYY should also be used in any situation where an implicit conversion will occur. Specifically:

- assigning a DATE variable to a CHAR variable, or vice versa
- passing an actual parameter of datatype DATE to a PL/SQL program unit whose corresponding formal parameter is of a character datatype, or vice versa.

4. Any Form parameters of datatype DATE must be eliminated. One possible strategy is to convert them to CHAR parameters that will hold a string representation of a date (with a 4-digit year) and to modify the application's logic accordingly.

5. Set the NLS_DATE_FORMAT environment variable to a value containing YYYY.

6. The effective Database format mask must never contain the YY format mask element. Instead, use one of the two following strategies:

- Use RR instead of YY. To accomplish this, you must alter your database session's NLS_DATE_FORMAT to a value containing RR in the PRE-FORM trigger of the initial form (and any form in the application that is invoked via an OPEN_FORM that specifies the SESSION option). For example:

```
FORMS_DDL(‘ALTER SESSION SET NLS_DATE_FORMAT =
            ‘’DD-MON-RR’’’);
```

Any SQL statements that implicitly convert a date to a string or vice versa (using the Database format mask) must be analyzed to see if they can potentially manipulate dates in the first half of the twentieth century (1900-1949). Any offending SQL statements should be modified to do a conversion with an explicit format mask containing RRRR or YYYY.

- Use YYYY instead of YY. This has already been accomplished in suggestion 4 above by setting NLS_DATE_FORMAT to a value containing YYYY (assuming of course that your application does not use ALTER SESSION to change that NLS_DATE_FORMAT setting).

  You must modify your application's logic to accommodate this change in the Database format mask from a 2-digit to a 4-digit year. For example, if you fetch a date column into a CHAR item without specifying an explicit format mask for that item, the item will now contain a string with a 4-digit year rather than a 2-digit year.

7. In the application's SQL code (in database queries), all TO_DATE and TO_CHAR constructs that specify a format mask containing YY must be changed to use RR or YYYY. YYYY is necessary if the dates being manipulated can potentially fall within the first half of the twentieth century. In this case, the application's logic must be modified accordingly.

8. If your application uses any positive or negative "infinity" dates (artificial dates that are meant to be higher or lower than any normal date your application might manipulate), make sure that they will still work properly in the 21st century. (So, for example, if your application uses Dec 31, 1999 as a positive infinity date, this will need to be corrected). The recommended positive infinity date is the highest date representable by the Oracle DATE datatype in all releases of Oracle Developer; namely, Dec 31, 4712. The recommended negative infinity date is Jan 1, 100 A.D. You can code these (for example) as:

```
TO_DATE('4712/12/31','YYYY/MM/DD') and
TO_DATE('0100/01/01','YYYY/MM/DD').
```

Any positive or negative infinity dates stored in your database should also conform to this guideline.

9. The obsolete datatypes EDATE and JDATE are *not* year-2000 compliant. Any application that defines items of those datatypes must be converted to use the DATE datatype instead. More specifically, these items must be made DATE4 items, as explained in suggestion 1 above.

0.1.3.2 Suggestions for Reports Applications

These suggestions apply to the non-date-enhanced releases of Report Builder: Releases 2.5.4.0.8, 2.5.5.2.5, 2.5.5.2.7, 2.5.7.2, and 2.5.7.3.

0.1.3.2.1 Areas to Consider for Year 2000 Compliance in Reports

In moving towards Year 2000 compliance, you should carefully analyze the following areas in your reports (both new and existing).

1. NLS_DATE_FORMAT environment variable

How do problems occur with NLS_DATE_FORMAT?

As explained earlier, the setting of the NLS_DATE_FORMAT environment variable affects the behavior of several areas. Unless this value is explicitly controlled, a 2-digit default will be used and can have a wide effect.

General suggestions for setting NLS_DATE_FORMAT

Explicity set the NLS_DATE_FORMAT environment variable, and set it to a value containing the 4-digit YYYY. This avoids the default 2-digit year format.

Also set the NLS_LANG environment variable. This ensures that the NLS-DATE_FORMAT value is used. (If necessary, you can override this setting on a report-by-report basis by using the ALTER SESSION mechanism.)

If this single explicit mask value is not appropriate for any particular date objects in the application, you can create individual date format masks for them.

2. PL/SQL

How do problems occur in PL/SQL?

Date-to-string and string-to-date conversions can occur in PL/SQL explicitly or implicitly. Year 2000-related problems can arise if the date format mask used to do the conversion is incorrect.

Explicit format mask conversions occur through calls to TO_DATE and TO_CHAR that specify an explicit format mask. These conversions are relatively easy to identify. YY format masks will probably lead to incorrect results. The RR mask may also cause problems.

Examples:

```
ch_var := TO_CHAR(date_var, ’DD-MON-YY’);
date_var := TO_DATE(’12 Feb 01’, ’DD-MON-YY’);
date_var := TO_DATE('12 Feb 01', 'DD-MON-RR’');
```

Implicit format mask conversions occur in several circumstances:

- TO_DATE and TO_CHAR function calls with no format masks specified
- assignment of a date variable to character variable or vice versa
- Passing in a date variable as a parameter when a character is expected
- Passing in a character variable as a parameter when a date is expected.

Examples:

```
ch_var := date_var;
date_var := ch_var;
date_var := TO_DATE('12 Feb 01');
```

These implicit conversions are less obvious and potentially more work to fix. In PL/SQL V1, implicit date-to-string and string-to-date conversions always use the DD-MON-YY format mask -- *regardless* of the NLS_DATE_FORMAT or NLS_LANG or NLS_TERRITORY settings. In other words, the NLS date format mask is ignored.

General suggestions for PL/SQL

Date strings in PL/SQL that are created using a 2-digit YY format must be fixed for Year 2000 compliance. There are two approaches to doing this:

- Decide on a "canonical" internal date format and use this consistently with internal date values. In general, for internal dates it is useful to preserve dates at the finest level of precision. A canonical date format of YYYY/MM/DD HH24:MI:SI is recommended. Adopting this may require that you increase character variable and column widths.
- Convert YY date format masks to RR. However, this approach is only feasible if the report manipulates dates exclusively within the 1950-2049 date range. The advantage of this approach (when it is feasible) is that it does not require expanding the widths of variables and columns.

  Note: Some releases of PL/SQL V1 used by the non-date-format enhanced releases of Reports do *not* support RR and RRRR masks. Therefore, if you want to use an RR mask in PL/SQL in those circumstances, you need to code a SELECT statement to do the conversion. For example:

  To fix

  ```
  date_var := TO_DATE(char_var, ‘DD-MON-YY’)
  ```

you need

```
SELECT TO_DATE(char_var, ‘DD-MON-RR’) into date_var from sys.dual;
```

Regardless of masks chosen, avoid unnecessary conversions between dates and strings as much as possible. Do not perform date comparisons or arithmetic using non-date types.

Avoid implicit conversions altogether. Specify all TO_DATE and TO_CHAR conversions with the appropriate mask. Although this is more work, the mask is saved with the report. (Within PL/SQL, localization of the masks is typically not important.)

3. SQL

This section applies to SQL in Reports queries as well as in PL/SQL.

How do problems occur in SQL?

Date-to-string and string-to-date conversions may occur in SQL explicitly or implicitly. Year 2000 problems can arise if the date format mask used to do the conversion is incorrect.

Explicit format mask conversions occur through calls to TO_DATE and TO_CHAR that specify an explicit format mask. YY format masks are the main causes of problems here. RR format masks may also cause problems if dates outside the 1950-2049 year range are being manipulated. Errors in explicit format mask conversions are relatively easy to identify while scanning the code (either manually or programmatically).

Errors introduced through implicit format mask conversions are less obvious. Implicit format mask conversions occur in several circumstances:

- TO_DATE and TO_CHAR function calls with no format masks specified
- expressions containing date and character types (e.g., hiredate < '01 Jan 85')
- Within PLSQL, selecting a date column into a character variable or column.

In SQL statements, implicit format mask conversions use the current Database format mask. This may be specified explicitly through the ALTER SESSION command (in a Report trigger). More typically, it is inherited from the NLS Date Format (on the client side). Year 2000 errors are possible if a 2-digit year is used for the Database format mask.

General suggestions for SQL

For Reports queries:

- Since Reports fully supports columns of type DATE for database, summaries, formulas and placeholders, usually there is no reason to convert a date column. In other words, if a column contains a date value, do not declare that column to be a character type. If you *must* convert a DATE column to a character column, specify a full format mask (DD-MON-YYYY HH24:MM:SS) to preserve maximum accuracy.

- Avoid expressions with mixed types. Avoid implicit conversions; instead, use TO_CHAR and TO_DATE and always specify the full canonical format mask. (However, check the appropriate edition of the *SQL Reference Manual* for the rules on implicit conversion; these rules are subject to change.)

For SQL within PLSQL :

- If you issue a SELECT <col>,.. INTO <var>,.. from PLSQL, avoid selecting a date column into a character column (<var>), and vice versa. If this is unavoidable, explicitly specify a TO_CHAR or TO_DATE with a canonical mask to convert the database column to the same type as the target column or variable. Note that a target column or character variable may need to have its width increased to accommodate a larger (more accurate) date string.

If for some reason you need to rely on implicit type conversions with SQL, set the database format mask to a 4-digit YYYY format mask if the NLS Date Format Mask is not appropriate (e.g., if RR). This can be done through the ALTER SESSION command in the Before Parameter Trigger.

4. Value If Null

Any reports column (excluding parameters) can have a Value If Null property set. This section applies to DATE columns with the Value If Null property set.

In current versions of Oracle Developer (Releases 1.x and 2.0), it is not possible in Report Builder to specify a format mask for the Value If Null setting (which is a character string).

How do problems occur with Value If Null?

By default, the format mask applied to a DATE column with a Value If Null is the NLS Date Format. If this evaluates to a 2-digit year format mask, then the Value If Null setting may be evaluated incorrectly.

General suggestions for Value If Null

Ensure that the effective format mask used in this case is a 4-digit year mask at a minimum. Setting the NLS Date Format appropriately is one way of ensuring this. Date-enhanced versions of Reports support another mechanism via the NVL_DATE_FORMAT.

5. Formatted Date Values

This section refers to date values formatted in the Reports output (to file, mail, printer, screen or previewer).

How do problems occur with formatted date values?

There are two sources of problems with fields that get their values from DATE columns:

- Date values are formatted using either the field-specific Format Mask, or (if no mask is specified for that field) the NLS Date Format Mask. If the effective format mask uses a 2-digit year, then the formatted results may be incorrect.
- Implicit type conversions can occur if you use the SRW.SET_FIELD_DATE builtin. For example, the call SRW.SET_FIELD_DATE(‘13 Feb 01’); will produce a character-to-date conversion.

General suggestions for formatted date values

In all cases, avoid using the YY format mask element.

Ensure that the effective format mask uses YYYY, if possible. Remember that switching to a 4-digit-year mask may require fields to be widened. If dates are guaranteed to be within the 1950-2049 year interval, the RR mask may be used to avoid changing the layout.

Independent of Year 2000 considerations, it is preferable to control date value formatting for layout fields by the NLS Date Format Mask rather than field-specific format masks. This facilitates localization. The implicit conversion in SRW.SET_FIELD_DATE calls should be avoided by specifying a DATE value as shown in this example (note the explicit format mask).

```
SRW.SET_FIELD_DATE(TO_DATE('2001/01/13', 'YYYY/MM/DD'));
```

RR/RRRR masks do not work in non-date-enhanced versions within PL/SQL.

If reports outputs (e.g., character mode report outputs) are processed further by software programs, it is necessary to use format masks of the required accuracy.

6. Parameters in the Runtime Parameter Form

DATE user parameters allow the specification of an Input mask that is used to convert the character string that the user specifies on a runtime parameter form or command line to a date.

How do problems occur in these parameters?

If no Input mask is specified for a DATE user parameter, the NLS Date Format Mask is used. If the effective format mask uses a 2-digit year format, the parameter value may be incorrect.

General suggestions for parameters

It is important to specify a 4-digit YYYY format mask to avoid loss of information.

In the report, if an Initial Value is specified for a DATE parameter, it is best if a 4-digit year is used here as well. Since this string is shown (as is) on the Runtime parameter form, specifying a 4-digit year in the Initial Value serves to remind the user that a 4-digit year is required.

The Input mask mechanism is useful to enforce a format mask per parameter. However, extensive use of this mechanism makes the Report hard to localize. It is preferable to control DATE user parameters by setting the NLS Date Format.

7. Infinity dates

If your application uses any positive or negative "infinity" dates (artificial dates that are meant to be higher or lower than any normal date your application might manipulate), make sure that they will still work properly in the 21st century. (So, for example, if your application uses Dec 31, 1999 as a positive infinity date, this will need to be corrected). The recommended positive infinity date is the highest date representable by the Oracle DATE datatype in all releases of Oracle Developer; namely, Dec 31, 4712. The recommended negative infinity date is Jan 1, 100 A.D. You can code these (for example) as:

TO_DATE('4712/12/31','YYYY/MM/DD') and
TO_DATE('0100/01/01','YYYY/MM/DD').

Any positive or negative infinity dates stored in your database should also conform to this guideline.

0.1.3.2.2 Additional Suggestions for New Reports Applications

If you are designing and coding a new application using a non-date-enhanced release of Report Builder, also do the following:

1. To facilitate changes and localization, avoid setting individual format masks for individual fields. (You can cause all date fields to use the same format mask through setting NLS_DATE_FORMAT). If a format mask must be specified for individual fields, use either the RR or YYYY format mask element. Never use the YY format mask element.
2. Specify YYYY in any Value If Null settings for columns (summaries, placeholders, formulas, and database).
3. In PL/SQL coding, select a canonical date format with the highest precision possible. A format mask of YYYY/MM/DD HH24:MI:SS is recommended. (Never use YY.)
4. Specify an explicit YYYY mask in all TO_CHAR and TO_DATE calls in PL/SQL and SQL statements.
5. Avoid coding a PL/SQL statement or a database query that could cause an implicit conversion from a date to string, or vice versa. Remember that if implicit conversions do

occur, they will be controlled by the NLS date format value, which might not be appropriate for all cases.

0.1.3.3 Suggestions for Graphics Applications

If you are using Graphics Builder Release 1.6 (the non-date-format-enhanced release) to either modify an existing application or create a new one, follow these suggestions:

1. **Query SQL/where clause**

   In a select statement having a TO_CHAR function of date or a TO_DATE function of char, use a format specification incorporating YYYY.

2. **Graphics data table internal display**

   Use a format specification incorporating YYYY, especially if using a discrete axis for dates.

3. **Discrete axis tick labels**

   Use the query property date format to control the appearance of date information. Use a format specification incorporating YYYY.

4. **Date axis tick labels**

   Use a format specification incorporating YYYY.

5. **Date axis min/max/step value**

   Because the date axis dialog allows no format other than DD-MON-YY, a work-around such as the following is suggested.

   Use built-ins to set the date value at either design time or runtime. For example, the following code could be executed before updating a chart axis when the user wants to specify a min/max/step:

   ```
   og_set_date_automin (chart_y_axis_hdl, false,
    to_date('2001/01/23', 'YYYY/MM/DD'));
   ```

   ```
   ... max...
   ... step...
   ```

6. **Date chart element labels**

   Use the menu option “format/date format” for changing.

   Use a format specification incorporating YYYY.

7. **Date parameter controlled via the display property**

   Use a format specification incorporating YYYY.

8. **Date property of an object**

Use CHAR properties for storing dates.

9. **Text objects**

If text objects are being used to display date information, it may be desirable to expand the width of fixed text objects and rearrange all text objects.

In the Designer, resize fixed text objects to accommodate a YYYY value, or convert fixed objects to expandable objects. This conversion can be done using the menu option “format/drawing option/text/ drawing option.” Unchecking the “fixed” checkbox makes the object expandable.

10. **PL/SQL**

- Instead of to-char (<date>),
  use to_char (<date>, ‘YYYY/MM/DD’).
- Instead of to_date (‘01-JUN-2003’),
  use to_date (‘2003/06/01’, ‘YYYY/MM/DD’).
- Instead of to_date (‘01-JUN-2003’, ‘DD-MON-YY’),
  use to_date (‘2003/06/01’, ‘YYYY/MM/DD’).
- Instead of char_variable := sysdate,
  use char_variable := to_char (sysdate, ‘YYYY/MM/DD’).

You could use the following debug interpreter commands to help modify your existing PL/SQL. (This would collect all the PL/SQL into one file.)

```
.export prog * file <filepath>
```

You could then use a text editor to locate any of the above constructs.

0.1.4 Year 2000 Suggestions for Developer Applications -- Enhanced Releases

To get maximum value from the following material, you may wish to first read the earlier general section on how dates are handled in the enhanced releases of Oracle Developer.

0.1.4.1 Suggestions for Forms Applications

These suggestions are for the date-format-enhanced releases of Form Builder: Releases 4.5.7.18.0, 4.5.10, 5.0.6, and 6.0.x.

The following suggestions are divided into those for modifying existing applications and those for creating new applications.

For definitions of the DATE2, DATE4, and DATETIME date objects, see the table on page -22. For the various date format masks used by Form Builder, see the descriptions starting on page -26.

0.1.4.1.1 Suggestions for the Build and Compile Environment

Whether you are creating a new application or working on an existing one, you should set the DATE variables for the environment in which Form Builder will operate during its design and compile stages.

Specifically, the FORMSnn_USER_DATE_FORMAT environment variable should be set to a format mask containing RRRR or YYYY. This ensures that centuries in dates (specified as the values of properties) will be preserved when you open and save an fmb file at design-time, and when you compile and generate an fmx file.

Form Builder uses the “input” and “output” format masks to convert dates entered into the property palette and to display dates in the property palette. See the discussions starting on page -30 for details on how the environment variables determine these format masks.

0.1.4.1.2 About Customized Forms

Not all of the suggestions in this white paper apply to forms that have been customized and are part of an Oracle Applications product. For more information, consult the Oracle Applications documentation or an Oracle Applications support representative.

0.1.4.1.3 Suggestions for New Forms Applications:

If you are designing and coding a new application, do the following to avoid creating Year 2000 problems:

1. Make a clear distinction in the application between "external" and “internal” date strings. External date strings are those visible to the end user. Internal date strings are those used as intermediate results, which ultimately produce date values. Internal date strings include strings produced by or passed to Forms builtins and PL/SQL implicit conversions.

   No single string should play both roles (for example, be produced by a Forms builtin and subsequently displayed to the end user).

2. Establish a single canonical format mask for all internal date strings in the application. It must contain full century and time information. A format mask of 'YYYY/MM/DD HH24:MI:SS' is suggested. If the application intends to manipulate dates prior to 1 A.D., SYYYY should be used instead of YYYY. The canonical format mask is established by placing the following statements in the PRE-LOGON trigger of the application's initial form(s):

   ```
    SET_APPLICATION_PROPERTY(DATE_FORMAT_COMPATIBILITY_MODE,
               '5.0');
   SET_APPLICATION_PROPERTY(PLSQL_DATE_FORMAT,
               'YYYY/MM/DD HH24:MI:SS');
   SET_APPLICATION_PROPERTY(BUILTIN_DATE_FORMAT,
               'YYYY/MM/DD HH24:MI:SS');
   ```

   Also, the following statement should be placed in the POST_LOGON trigger of the application’s initial form(s), and also in the PRE-FORM trigger of any form invoked via an OPEN_FORM that specifies the SESSION option:

   ```
   FORMS_DDL('ALTER SESSION SET NLS_DATE_FORMAT =
          "YYYY/MM/DD HH24:MI:SS"');
   ```

   Note: A prior version of this white paper recommended placing these initialization statements in the PRE_FORM trigger. However, this may not be adequate if your initial form contains any of the following "early" triggers and they manipulate dates:

- PRE-LOGON, ON-LOGON, or POST-LOGON
- WHEN-CREATE-RECORD for a block whose Single Record property is Yes (this applies only to Forms 5.0 and above).

If the initial form does not contain any of these "early" triggers, or if they exist but do not manipulate dates, then placing the initialization statements in the PRE-FORM trigger is an acceptable alternative to placing them in the PRE-LOGON and POST-LOGON triggers. Also note that it does not hurt to place them in more than one location.

The crucial point is that the initialization of the PLSQL, Builtin, and Database format masks must occur before they are used.

3. Never use an explicit format mask when converting an internal date string to or from a date.

An exception can be made for hard-coded dates. For example, you might code TO_DATE('1900/01/01', 'YYYY/MM/DD') rather than TO_DATE('1900/01/01 00:00:00').

4. In general, use the RRRR element in any format masks used for external date strings. These format masks include:

- Explicit format masks for text items and display items of datatype DATE or DATETIME.
- Default format masks for text items, display items, and LOVs of datatype DATE or DATETIME. These are derived from environment variables such as:

  FORMSnn_USER_DATE_FORMAT,
  FORMSnn_USER_DATETIME_FORMAT,
  FORMSnn_OUTPUT_DATE_FORMAT,
  FORMSnn_OUTPUT_DATETIME_FORMAT, and potentially
  NLS_DATE_FORMAT.
- Format masks specified in TO_CHAR, used to produce a string that is ultimately displayed in a message, LOV, or CHAR item.
- Format masks specified in TO_DATE, used to derive a date from a string that was ultimately derived from a CHAR item.

A format mask element of RRRR is preferable to YYYY because it facilitates end user input: For YYYY, a 2-digit year is interpreted as representing a date in the first century A.D, whereas for RRRR, a 2-digit year is interpreted as representing a date in the range 1950-2049.

In applications where screen real estate is at a premium, you might consider using RR instead of RRRR for items guaranteed to contain dates in the range 1950-2049. The potential savings in screen real estate must be weighed against the potential for end user confusion.

5. Also consider the advice below in suggestion 7 for the canonical technique.

0.1.4.1.4 Suggestions for Existing Forms Applications:

Because the date-format-enhanced releases of Form Builder are only recently available, almost all existing applications will have been developed using an earlier release. These applications can be re-compiled using a later, date-format-enhanced release. However, this will not automatically solve all Year 2000 problems. You still may find it necessary to make some modifications to the application. In many cases, the modifications will consist simply of adding a few statements to a PL/SQL trigger.

Modifications are required for any application that was developed using a non-date-format-enhanced release, and that does any of the following:

- Contains any DATE2 objects other than LOVs.
- Uses a format mask containing YY, other than in a TO_CHAR for producing a string used purely for display purposes.
- Uses a TO_CHAR (on a date) without a format mask, or does an implicit PL/SQL conversion from a date to a string, unless the resulting string is used purely for display purposes.
- Uses a TO_DATE without a format mask, or does an implicit PL/SQL conversion from a string to a date.

Choice of two modification techniques

Modifications to an application for Year 2000 compliance should employ one of two techniques:

- either the "canonical" technique
- or the "RR" technique.

Note: If you are modifying a customized form in an Oracle Applications product, you must use the RR technique.

The "canonical" technique

This technique is simply to modify the application so that it adheres to the suggestions for new applications. It’s called the "canonical" technique because it involves establishing a single canonical format mask for all internal date strings.

Some applications may require no modification other than adding some statements to establish the canonical format mask (suggestion number 2 for new applications). But other applications may require additional modifications. which can vary from the minor to the extensive.

The "RR" technique

This technique is essentially to substitute RR for YY and RRRR for YYYY in all format masks used by the application.

The RR technique cannot be used for applications that manipulate dates prior to 100 A.D. It is best suited to applications that do not manipulate dates prior to 1950.

Comparison of the canonical and RR techniques

When resources for Year 2000 work are limited, you will probably want to choose the technique that entails less effort. If an application’s DATE2 objects never store dates in the first half of the twentieth century, and its date strings with 2-digit years never represent such dates, then the RR technique will probably be easiest.

However, there are long-term advantages to choosing the canonical technique:

- Explicit format masks (and explicit TO_DATE and TO_CHAR conversions) can be eliminated in many cases, resulting in more readable and less error-prone code. For example, the result of NAME_IN (or any other builtin that returns a string representation of a date value) can be directly assigned to any date object.
- NAME_IN, COPY, and other builtins can be used to assign values from any date object to any other date object. (This is not always possible with the RR technique; some releases impose restrictions on using these builtins on combinations of unlike date objects.)

After you have chosen the technique you will use, follow the appropriate set of steps below.

Steps to implement the canonical technique

(The following suggestions assume the application is designed to operate correctly prior to the year 2000.)

1. You must analyze the usage of every string produced from a date by an operation that uses the Database, PLSQL, or Builtin format mask, as follows:

- Any string produced from a date by an operation that uses the Database or PLSQL format mask (an implicit SQL or PL/SQL conversion or a TO_CHAR without a format mask) must not be displayed to the end user (in a message or in a CHAR item).

  Instead, a TO_CHAR with an explicit format mask should be used. The correct format mask will typically be the one given by GET_APPLICATION_PROPERTY(OUTPUT_DATE_FORMAT).

- Any string which is produced from a date by an operation that uses the Builtin format mask must not be displayed to the end user (in a message or in a CHAR item).

  Instead, it should be converted to a date (by using TO_DATE without a format mask or by assigning it to a local DATE variable) and then converted to a string as in point "a" above.

- If a string produced from a date by an operation that uses the Database, PLSQL, or Builtin format mask is compared with a literal string or analyzed by string operations (such as SUBSTR), then the logic of the application will require modification (to reflect the fact that the string is now in canonical format).

2. Similarly, you must analyze the logic that produces any string which is subsequently used to produce a date via an operation that uses the Database, PLSQL, or Builtin format mask, as follows:

- No string entered by the end user (in a CHAR item) may be used to produces a date via an operation using the Database or PLSQL format mask (an implicit SQL or PL/SQL conversion or a TO_CHAR without a format mask).

  Instead, a TO_DATE with an explicit format mask should be used to convert the value in the CHAR item (entered by the end user). The correct format mask will typically be the one given by GET_APPLICATION_PROPERTY(USER_DATE_FORMAT). More complicated logic will be required if your installation standards allow the FORMSnn_USER_DATE_FORMAT environment variable to be set to a value containing multiple format masks separated by vertical bars.

- No string entered by the end user (in a CHAR item) may be used to produce a date via an operation that uses the Builtin format mask.

Instead, the value in the CHAR item (entered by the end user) should be converted to a date using a TO_DATE with an explicit format mask, as in point "a" above. In some cases, the resulting date value can be used directly as input to the Builtin operation. In other cases (where the PL/SQL compiler cannot unambiguously determine that an implicit date-to-string conversion is appropriate) it will be necessary to explicitly convert the date value to a string (by using TO_CHAR without a format mask or by assigning it to a local CHAR variable).

- If a string that produces a date via an operation using the Database, PLSQL, or Builtin format mask is a literal string or assigned from a literal string, or built by string operations (such as CONCAT), then the logic of the application will require modification (to reflect the fact that the string should be in canonical format).

3. Local PL/SQL variables that are used to hold internal date strings may need to be enlarged.

4. If an explicit format mask is used to convert an internal date string to or from a date, that explicit format mask should be removed. (Reminder: an internal date string is one used as an intermediate result, which ultimately produces a date value.)

An exception can be made for hard-coded dates. For example, you might code TO_DATE(‘1900/01/01’, ‘YYYY/MM/DD’) rather than TO_DATE(‘1900/01/01 00:00:00’).

5. Format masks used for external string-to-date conversions must be analyzed to see if any contain the YY format mask element. (Reminder: an external string is one visible to the end user.) These format masks include:

- Explicit format masks for text items of datatype DATE or DATETIME.
- Default format masks for text items of datatype DATE or DATETIME.

  These are derived from environment variables such as FORMSnn_USER_DATE_FORMAT, FORMSnn_USER_DATETIME_FORMAT, and potentially NLS_DATE_FORMAT.
- Format masks specified in TO_DATE, used to derive a date from a string that was ultimately derived from a CHAR item.

A format mask element of YY is not permissible. It must be converted to RR, RRRR, or YYYY. Converting a YY format mask element to RR is permissible only if the string being converted to a date is guaranteed never to represent a date in the first half of the twentieth century (1900-1949).

A format mask element of RRRR is preferable to YYYY because it eases end user input. For YYYY, a 2-digit year is interpreted as representing a date in the first century A.D, whereas for RRRR, a 2-digit year is interpreted as representing a date in the range 1950-2049.

Bear in mind that converting YY to RRRR or to YYYY in an item's format mask will increase the effective value of the item's Maximum Length property. (It will be increased at runtime to accommodate the maximum length string that can be produced by the format.) You may wish to increase the item's Width property so that the entire value is visible. Of course, this may require that adjacent items be moved.

6. The YY format mask element is permissible in format masks used purely for external date-to-string conversions. (In date-to-string conversions, YY is equivalent to RR, and YYYY is equivalent to RRRR). However, you may wish to convert these to RR (for consistency of style) or to RRRR or YYYY (to provide clearer output for the end user).

These format masks include:

- Explicit format masks for display items of datatype DATE or DATETIME.
- Format masks specified in TO_CHAR, used to produce a string which is ultimately displayed in a message, LOV, or CHAR item.

When converting YY to RRRR or to YYYY, bear in mind the points made in suggestion 5 about the effects on an item's Maximum Length and Width property.

7. If your application uses any positive or negative "infinity" dates (artificial dates that are meant to be higher or lower than any normal date your application might manipulate), make sure that they will still work properly in the 21st century. (Thus, for example, if your application uses Dec 31, 1999 as a positive infinity date, this will need to be corrected).

The recommended positive infinity date is the highest date representable by the Oracle DATE datatype in a date-format-enhanced release of Oracle Developer; namely, Dec 31, 9999. You can code this (for example) as:

```
TO_DATE('9999/12/31','YYYY/MM/DD')
```

In the canonical technique, the recommended negative infinity dates depend on whether your application manipulates dates prior to 1 A.D.

- If your application manipulates dates prior to 1 A.D., the recommended negative infinity date is the lowest date representable by the Oracle DATE datatype in a date-format-enhanced release of Oracle Developer; namely, Jan 1, 4712 B.C. You can code this (for example) as:

  ```
  TO_DATE('-4712/01/01','SYYYY/MM/DD').
  ```

- If your application does *not* manipulate dates prior to 1 A.D., the recommended negative infinity date is Jan 1, 1 A.D. For example:

TO_DATE('0001/01/01','YYYY/MM/DD')

Any positive or negative infinity dates stored in your database should also conform to this guideline.

8. The obsolete datatypes EDATE and JDATE are *not* year-2000 compliant. If your application contains items of those datatypes, they must be converted to use the DATE datatype (following previous suggestions to ensure year-2000 compliance).

Steps to implement the RR technique

(The following suggestions assume that the application is designed to operate correctly prior to the year 2000.)

1. Analyze all DATE2 items to see if any can potentially store dates in the first half of the twentieth century (1900-1949).

- If the only such dates are artificial negative infinity dates, change them to Jan 1, 1950.
- If a DATE2 item can store legitimate (non-infinity) dates prior to 1950, convert it to a DATE4 item. Also modify any logic that produces a string from such an item (e.g., using NAME_IN) or that sets the item's value from a string (e.g., using COPY) in order to accommodate the fact that the string will now contain a 4-digit year rather than a 2-digit year. Local PL/SQL variables that are used to hold such strings may need to be enlarged.

When converting YY to RRRR or YYYY in an item's Format Mask property and/or increasing its Maximum Length property, bear in mind the points made in suggestion 4 for the canonical technique, about the effects on an item's Maximum Length and Width property.

2. Similarly, analyze all strings that represent dates and that contain 2-digit years. If any can potentially store dates in the first half of the twentieth century (1900-1949), modify the application's logic so that a 4-digit year is used instead. This entails (for example) using format masks with 4-digit years to convert these strings to or from a date.

Hard-coded dates often violate this guideline. For example, consider TO_DATE('01-jan-20'). This should be converted to TO_DATE('1920/01/01', 'YYYY/MM/DD').

3. You must analyze all Form parameters of datatype DATE to see if any can potentially store dates in the first half of the twentieth century (1900-1949). If any are found, they must be removed. One possible strategy is to convert them to CHAR parameters which will hold a string representation of a date (with a 4-digit year), and to modify the application's logic accordingly.

4. You must add the following statements to the PRE-LOGON trigger of the application's initial form(s):

```
SET_APPLICATION_PROPERTY(PLSQL_DATE_FORMAT, 'DD-MON-RR');
SET_APPLICATION_PROPERTY(BUILTIN_DATE_FORMAT, 'RR');
```

Setting BUILTIN_DATE_FORMAT to 'RR' modifies the default Forms behavior so as to substitute RR for YY (and RRRR for YYYY) in the various different Builtin format masks used for DATE2, DATE4, and DATETIME objects.

Note 1: A prior version of this white paper recommended placing these initialization statements in the PRE_FORM trigger. However, this may not be adequate if your initial form contains any of the following "early" triggers and they manipulate dates:

- PRE-LOGON, ON-LOGON, or POST-LOGON
- WHEN-CREATE-RECORD for a block whose Single Record property is Yes (this applies only to Forms 5.0 and above).

If the initial form does not contain any of these "early" triggers, or if they exist but do not manipulate dates, then placing the initialization statements in the PRE-FORM trigger is an acceptable alternative to placing them in the PRE-LOGON trigger. Also note that it does not hurt to place them in more than one location.

The crucial point is that the initialization of the PLSQL and Builtin format masks must occur before they are used.

Note 2: The above suggestions should not be followed for a customized form in an Oracle Applications product. The relevant application properties are already being set in forms supplied by Oracle Applications.

5. If the Database format mask contains YY, it must be replaced by RR.

Therefore, if the NLS_DATE_FORMAT environment variable is being set to a value containing YY, it must be changed to use RR instead. If the NLS_DATE_FORMAT environment variable is *not* being set, you must determine the default value for your NLS environment. (For American, the default is DD-MON-YY). If this default value contains YY, you must set NLS_DATE_FORMAT to the corresponding value with YY replaced by RR, *and* you must also set NLS_LANG to ensure that the NLS_DATE_FORMAT environment variable will be used for Database mask conversions.

If your application contains any statements that alter your database session's NLS_DATE_FORMAT to a value containing YY, they too must be modified to use

RR.

6. Other format masks used for string-to-date conversions must be analyzed to see if any contain the YY format mask element. These format masks include:

   - Explicit format masks for text items of datatype DATE or DATETIME.
   - Default format masks for text items of datatype DATE or DATETIME.

     These are derived from environment variables such as FORMSnn_USER_DATE_FORMAT, FORMSnn_USER_DATETIME_FORMAT, and potentially NLS_DATE_FORMAT.
   - Format masks specified in TO_DATE.

   Each YY format mask element should be converted to RR.

7. The YY format mask element is permissible in format masks used purely for date-to-string conversions. (In date-to-string conversions, YY is equivalent to RR). However, you may wish convert these to RR for consistency of style.

   These format masks include:

   - Explicit format masks for display items of datatype DATE or DATETIME.
   - Format masks specified in TO_CHAR used for producing a string that is ultimately displayed in a message, LOV, or CHAR item.

8. If your application uses any positive or negative "infinity" dates (artificial dates that are meant to be higher or lower than any normal date your application might manipulate), make sure that they will still work properly in the 21st century. (Thus, for example, if your application uses Dec 31, 1999 as a positive infinity date, this will need to be corrected).

   The recommended positive infinity date is the highest date representable by the Oracle DATE datatype in a date-format-enhanced release of Oracle Developer; namely, Dec 31, 9999. You can code this (for example) as:

   ```
   TO_DATE('9999/12/31','YYYY/MM/DD')
   ```

   (If you intend to use a positive infinity date in a DATE2 object, then you should instead use TO_DATE(‘49/12/31’,’RR/MM/DD’).)

   When using the RR technique, the recommended negative infinity date is Jan 1, 100 A.D. For example:

   ```
   TO_DATE('0100/01/01','YYYY/MM/DD')
   ```

(If you intend to use a negative infinity date in a DATE2 object, then you should instead use TO_DATE(‘50/01/01’,’RR/MM/DD’).)

Any positive or negative infinity dates stored in your database should also conform to this guideline.

9. Given a string containing a 2-digit year, do not convert it to a date using a format mask containing YYYY unless it really represents a date in the first century A.D. This may seem obvious, but in fact this bug can appear is subtle ways that do not cause problems until the year 2000.

For example:

```
TO_CHAR(TO_DATE(rr_string, 'DD-MON-YYYY'), 'DD-MON-RR')
```

happens to work for RR dates (1950-2049) *except* for the year 2000 (which produces the error ORA-01841, because there is no year 0). And comparisons such as

```
TO_DATE(rr_string_1, 'DD-MON-YYYY') <
TO_DATE(rr_string_2, 'DD-MON-YYYY')
```

will work correctly if both dates are prior to 2000 or both are later than 2000, but not otherwise.

Unfortunately, in real applications instances of this bug are seldom as obvious as in the examples above. Often the result of the offending TO_DATE will be assigned to a local variable, and then later used in a TO_CHAR or in a comparison. And the offending format masks may not be explicitly specified. For example:

```
string1 := NAME_IN('date2_object_1');
string2 := NAME_IN('date2_object_2');
  -- NAME_IN uses DD-MON-RR on DATE2 objects
     ...
date1 := TO_DATE(string1, 'DD-MON-YYYY');
date2 := TO_DATE(string2, 'DD-MON-YYYY');
 -- date1 and date2 now contain dates in the first century A.D.
...
if date1 < date2 then -- erroneous comparison
...
end if;
```

10. The obsolete datatypes EDATE and JDATE are *not* year-2000 compliant. If your application contains items of those datatypes, they must be converted to use the DATE datatype (following previous suggestions to ensure year-2000 compliance).

0.1.4.2 Suggestions for Reports Applications

These suggestions apply to the date-enhanced releases of Report Builder: Releases 2.5.5.20, 2.5.7.4.3, 2.5.7.5, and 3.0.5.

0.1.4.2.1 Areas to Consider for Year 2000 Compliance in Reports

In moving towards Year 2000 compliance, you should carefully analyze the following areas in your reports (both new and existing).

Note that the setting of the NLS_DATE_FORMAT environment variable affects the behavior several of these areas.

1. NLS_DATE_FORMAT environment variable

How do problems occur with NLS_DATE_FORMAT?

As explained earlier, the setting of the NLS_DATE_FORMAT environment variable affects the behavior of several areas. Unless this value is explicitly controlled, a 2-digit default will be used and can have a wide effect.

General suggestions for setting NLS_DATE_FORMAT

Explicity set the NLS_DATE_FORMAT environment variable, and set it to a value containing the 4-digit YYYY or RRRR. This avoids the default 2-digit year format. (Remember that RRRR may not be appropriate in all situations.)

Also set the NLS_LANG environment variable. This ensures that the NLS-DATE_FORMAT value is used. (If necessary, you can override this setting on a report-by-report basis by using the ALTER SESSION mechanism.)

If this single explicit mask value is not appropriate for any particular date objects in the application, you can create individual date format masks for them.

2. PL/SQL

How do problems occur in PL/SQL?

Date-to-string and string-to-date conversions can occur in PL/SQL explicitly or implicitly. Year 2000-related problems can arise if the date format mask used to do the conversion is incorrect.

Explicit format mask conversions happen through calls to TO_DATE and TO_CHAR that specify an explicit format mask. Incorrect conversions are relatively easy to identify. YY format masks will probably lead to incorrect results.

Examples:

```
ch_var := TO_CHAR(date_var, 'DD-MON-YY');
date_var := TO_DATE('12 Feb 01', 'DD-MON-YY')
```

Implicit format mask conversions happen under several circumstances:

- TO_DATE and TO_CHAR function calls with no format masks specified.
- assignment of a date variable to character variable or vice versa.
- Passing in a date variable as a parameter when a character is expected.
- Passing in a character variable as a parameter when a date is expected.

Examples:

```
ch_var := date_var;
date_var := ch_var;
date_var := TO_DATE('12 Feb 01');
```

These implicit conversions are less obvious and potentially more work to fix. In PL/SQL V1, implicit date-to-string and string-to-date conversions default to use of a DD-MON-YY format mask -- *regardless* of the NLS_DATE_FORMAT or NLS_LANG or NLS_TERRITORY settings. In other words, the NLS date format mask is ignored. (But see the suggestion below for use of the PLSQL_DATE_FORMAT command to create a different implicit default.)

General suggestions for PL/SQL

Date strings in PL/SQL that were created using a 2-digit YY format must be fixed for Year 2000 compliance. There are two approaches to doing this:

- Decide on a "canonical" internal date format and use this consistently with "internal" date values. In general, for internal dates, it is useful to preserve dates at the finest level of precision. A canonical date format of YYYY/MM/DD HH24:MI:SS is recommended. Adopting this may require that character variable and column widths be increased.
- Convert YY date format masks to RR. However, this approach is only feasible if the Report manipulates dates exclusively within the 1950-2049 date range. The benefit of this approach is that it does not require expanding the widths of variables and columns.

After choosing between the canonical and RR approaches, choose one of the following ways of implementing the new mask:

- Avoid implicit conversions altogether. Specify all TO_DATE and TO_CHAR conversions with the appropriate mask. Although this is more work, the mask is saved with the report. (Within PL/SQL, localization of the masks is typically not important.)
- Establish the date format mask used in implicit PL/SQL type conversions by specifying PLSQL_DATE_FORMAT=<format_mask> on the command line. While easier to implement, this approach relies on a command line option being specified. (This is not stored persistently in the report.)

As far as possible, avoid unnecessary conversions between dates and strings. Do not perform date comparisons or arithmetic using non-date types.

3. SQL

This section applies to SQL in Reports queries as well as in PL/SQL.

How do problems occur in SQL?

Date-to-string and string-to-date conversions may occur in SQL explicitly or implicitly. Year 2000 problems can arise if the date format mask used to do the conversion is incorrect.

Explicit format mask conversions happen through calls to TO_DATE and TO_CHAR that specify an explicit format mask. Incorrect conversions are relatively easy to identify. YY format masks are the main causes of problems here. RR format masks may also cause problems if dates prior to 100 A.D are being manipulated.

Implicit format mask conversions happen under several circumstances:

- TO_DATE and TO_CHAR function calls with no format masks specified
- expressions containing date and character types (e.g., hiredate < '01 Jan 85')
- Within PLSQL, selecting a date column into a character variable or column.

In SQL statements, implicit conversions use the current Database format mask. This may be specified explicitly through the ALTER SESSION command (in a Report trigger). More typically, it is inherited from the NLS Date Format (on the client side). Year 2000 errors are possible if a 2-digit year is used for the database format mask.

General suggestions for SQL

For Reports queries:

- Since Reports fully supports columns of type DATE for database, summaries, formulas and placeholders, usually there is no reason to convert a date column. In other words, if a column contains a date value, do not declare that column to be a character type. If you *must* convert a DATE column to a character column, specify a full format mask (DD-MON-YYYY HH24:MM:SS) to preserve maximum accuracy.

- Avoid expressions with mixed types. Avoid implicit conversions; instead, use TO_CHAR and TO_DATE and always specify the full canonical format mask. (However, check the appropriate edition of the *SQL Reference Manual* for the rules on implicit conversion; these rules are subject to change.)

For SQL within PLSQL :

- If you issue a SELECT <col>,.. INTO <var>,.. from PLSQL, avoid selecting a date column into a character column (<var>), and vice versa. If this is unavoidable, explicitly specify a TO_CHAR or TO_DATE with a canonical mask to convert the database column to the same type as the target column or variable. Note that a target column or character variable may need to have its width increased to accommodate a larger (more accurate) date string.

If for some reason you need to rely on implicit type conversions with SQL, set the database format mask to a 4-digit YYYY format mask if the NLS Date Format Mask is not appropriate (e.g., if RR). This can be done through the ALTER SESSION command in the Before Parameter Trigger.

4. Value If Null

Any reports column (excluding parameters) can have a Value If Null property set. This section applies to DATE columns with the Value If Null property set.

In current versions of Oracle Developer (Releases 1.x and 2.0), it is not possible in Report Builder to specify a format mask for the Value If Null setting (which is a character string).

How do problems occur with Value If Null?

By default, the format mask applied to a DATE column with a Value If Null is the NLS Date Format. If this evaluates to a 2-digit year format mask, then the Value If Null setting may be evaluated incorrectly.

General suggestion for Value If Null

Ensure that the effective format mask used in this case is a 4-digit year mask at a minimum. Setting the NLS Date Format appropriately is one way of ensuring this.

Date-enhanced versions of Reports support another mechanism via the keyword NVL_DATE_FORMAT=<format_mask> on the command line. However, this approach relies on a command line action, and the value is not stored permanently in the report.

5. Formatted Date Values

This section refers to date values formatted in the Reports output (to file, mail, printer, screen or previewer).

How do problems occur with formatted date values?

There are two sources of problems with fields that get their values from DATE columns:

- Date values are formatted using either the field-specific Format Mask, or (if no mask is specified for that field) the NLS Date Format Mask. If the effective format mask uses a 2-digit year, then the formatted results may be incorrect.
- Implicit type conversions can occur if you use the SRW.SET_FIELD_DATE builtin. For example, the call SRW.SET_FIELD_DATE(‘12 Feb 01’); will produce a character-to-date conversion.

General suggestions for formatted date values

In all cases, avoid using the YY format mask element.

Ensure that the effective format mask uses a 4 digit year mask if possible. The YYYY format should be used if dates before 100 AD are possible. This may require fields to be widened. If dates are guaranteed to be within the 1950-2049 year interval, the RR mask may be used to avoid changing the layout.

Independent of Year 2000 considerations, it is preferable to control date value formatting for layout fields by the NLS Date Format Mask rather than field-specific format masks. This facilitates localization. The implicit conversion in SRW.SET_FIELD_DATE calls should be avoided by specifying a DATE value as shown in this example (note the explicit format mask).

```
SRW.SET_FIELD_DATE(TO_DATE('2001/02/13', 'YYYY/MM/DD'));
```

If reports outputs (e.g., character mode report outputs) are processed further by software programs, it is necessary to use format masks of the required accuracy.

6. Parameters in the Runtime Parameter Form

DATE user parameters allow the specification of an Input mask that is used to convert the character string that the user specifies on a runtime parameter form or command line to a date.

How do problems occur in these parameters?

If no Input mask is specified for a DATE user parameter, the NLS Date Format Mask is used. If the effective format mask uses a 2-digit year format, the parameter value may be incorrect.

General suggestions for parameters

It is important to specify a 4-digit format mask to avoid loss of information. A format mask of RRRR (rather than YYYY) allows the user to type in just a 2-digit year, which is friendlier from the user's perspective.

In the report, if an Initial Value is specified for a DATE parameter, it is best if a 4-digit year is used here as well. Since this string is shown (as is) on the Runtime parameter form, specifying a 4-digit year in the Initial Value serves to remind the user that a 4-digit year is required.

The Input mask mechanism is useful to enforce a format mask per parameter. However, extensive use of this mechanism makes the Report hard to localize. It is preferable to control DATE user parameters through the NLS Date Format.

7. Infinity dates

If your application uses any positive or negative "infinity" dates (artificial dates that are meant to be higher or lower than any normal date your application might manipulate), make sure that they will still work properly in the 21st century. (Thus, for example, if your application uses Dec 31, 1999 as a positive infinity date, this will need to be corrected).

The recommended positive infinity date is the highest date representable by the Oracle DATE datatype in a date-format-enhanced release of Oracle Developer; namely, Dec 31, 9999. You can code this (for example) as:

    TO_DATE('9999/12/31','YYYY/MM/DD')

The recommended negative infinity date depends on what dates your application processes.

- If your application does not manipulate dates prior to 100 A.D., the recommended negative infinity date is Jan 1, 100 A.D. You can code this (for example), as TO_DATE('0100/01/01', 'YYYY/MM/DD').
- If your application does manipulate dates prior to 100 A.D., the recommended negative infinity date is the lowest date representable by the Oracle DATE datatype in a date-format-enhanced release of Oracle Developer; namely, Jan 1, 4712 B.C. You can code this (for example) as: TO_DATE('-4712/01/01','SYYYY/MM/DD').

  Note that B.C. data must be manipulated using the SYYYY rather than YYYY or RRRR mask elements.

Any positive or negative infinity dates stored in your database should also conform to the guidelines you choose.

8. Potential problems with conversions

Given a string containing a 2-digit year, do not convert it to a date using a format mask containing YYYY unless it really represents a date in the first century A.D. This may seem obvious, but in fact this bug can appear is subtle ways that do not cause problems until the year 2000.

For example:

```
TO_CHAR(TO_DATE(rr_string, 'DD-MON-YYYY'), 'DD-MON-RR')
```

happens to work for RR dates (1950-2049) *except* for the year 2000 (which produces the error ORA-01841, because there is no year 0). And comparisons such as

```
TO_DATE(rr_string_1, 'DD-MON-YYYY') <
TO_DATE(rr_string_2, 'DD-MON-YYYY')
```

will work correctly if both dates are prior to 2000 or both are later than 2000, but not otherwise.

Unfortunately, in real applications instances of this bug are seldom as obvious as in the examples above. Often the result of the offending TO_DATE will be assigned to a local variable, and then later used in a TO_CHAR or in a comparison. And the offending format masks may not be explicitly specified.

0.1.4.2.2 Additional Suggestions for New Reports Applications

If you are designing and coding a new application using a date-enhanced release of Report Builder, also do the following:

1. In PL/SQL coding, select a canonical date format with the highest precision possible. A format mask of YYYY/MM/DD HH24:MM:SS is recommended. (Never use the 2-digit YY format mask element.)
2. To facilitate changes and localization, avoid setting individual format masks for individual fields when it is unnecessary. (You can cause all date fields to use the same format mask through the NLS date format setting.) If a format mask must be specified for

individual fields, use either the RR, RRRR or YYYY format mask element. Never use the YY format mask element.

3. Specify a 4-digit year in any Value If Null settings for columns (summaries, placeholders, formulas and database). If the NLS date format value is not appropriate for the Value If Null settings, use the NVL_DATE_FORMAT command line keyword to override that value.

4. Avoid coding a PL/SQL statement or a database query which could cause an implicit conversion from a date to string, or vice versa. Remember that if implicit conversions do occur, they will be controlled by the NLS date format value, which might not be appropriate for all cases.

5. Specify an explicit mask for all TO_CHAR and TO_DATE calls in PL/SQL and SQL statements.

0.1.4.3 Suggestions for Graphics Applications

In contrast to Form Builder and Report Builder, the treatment of dates in Graphics Builder has not changed in its recent releases. Therefore, the Year 2000 compliance suggestions offered above for Graphics in the non-date-enhanced section of this paper apply to all releases of Graphics.

Later releases of Graphics do use later versions of PL/SQL, which support the RR and RRRR date formats. However, for Graphics applications, the YYYY format is preferred wherever possible.

0.1.5 Appendix: Applications that Use Other Data Sources

When accessing a non-Oracle data source using ODBC, or accessing an Oracle Lite or Rdb data source, date processing in Developer is restricted. In general, it is recommended that you avoid any date-to-string or string-to-date conversion within queries and when fetching, inserting, or updating data.

Not all conversions are allowed on all data sources, and when conversions are allowed, they typically do not use the NLS date format. (The NLS date format is established via Oracle environment variables.) Instead, conversions often use a format mask that conforms to the ODBC standard, namely one of these three:

```
YYYY-MM-DD HH24:MI:SS[.FFFFFF]

YYYY-MM-DD

HH24:MI:SS[.FFFFFF]
```

where [.FFFFFF] represents an optional variable-length fractional seconds part.

If your application requires specialized processing that would be facilitated by doing any of the date-to-string or string-to-date conversions mentioned above, you must determine whether they are supported for your data source, and also which format masks they use. You may need to write specialized logic to manipulate strings on the client side.

For example, Oracle date format masks do not support a fractional portion. So if you wished to have a Forms trigger convert a string in the first format listed above into a DATETIME item, you would have to code something like:

```
:item := TO_DATE(SUBSTR(string, 1, 19), 'YYYY-MM-DD HH24:MI:SS');
```

Bear in mind that doing any of the date-to-string or string-to-date conversions mentioned above may compromise the portability of your application to other data sources.

Also note that ODBC does not support ALTER SESSION.

Oracle Lite:

Oracle Lite versions 3.0 and above support Oracle-style to_char and to_date functions within SQL statements. Also, Oracle Lite does use the format mask given in the NLS_DATE_FORMAT parameter within the polite.ini file (if present) for to_char and to_date functions and for implicit conversions. (If this parameter is absent, the default ODBC format is used. Refer to your Oracle Lite documentation for more details.)

Because Oracle Lite does support NLS_DATE_FORMAT, you can set that parameter as recommended elsewhere in this white paper. However, because Oracle Lite (like other ODBC data sources) does *not* support ALTER SESSION, you cannot fully implement the suggestions for the canonical technique.

You can work around the absence of ALTER SESSION by doing all of the following:

- explicitly specify the canonical format mask in all to_date and to_char conversions between dates and strings in SQL statements
- avoid selecting date columns into string variables without an explicit conversion
- avoid selecting string columns into date variables without an explicit conversion
- avoid inserting or updating date columns from string variables without an explicit conversion
- avoid inserting or updating string columns from date variables without an explicit conversion.