

TS-566D

***Processing ORACLE® Dates with SAS/ACCESS® in
the UNIX® Environment***

***Technical Support Division
UNIX Systems Interface***

***SAS Institute, Inc.
SAS Campus Drive
Cary, N.C. 27513***

Abstract

This paper is meant to document and clarify the issues involved when dealing with date conversion for ORACLE on UNIX platforms. Due to the increased interest of the SAS/ACCESS Interface to ORACLE product in the UNIX environment, this has been a popular topic for a number of SAS users.

Introduction

The SAS/ACCESS Interface to ORACLE product provides support for the SQL Procedure Pass-Through Facility, the ACCESS procedure and the DBLOAD Procedure. With SAS Release 7.x, the LIBNAME engine support will be added to the SAS/ACCESS product and the date conversion issues discussed here will also follow the same rules.

There are varying data types for different Database Management Systems (DBMSs) for which the SAS/ACCESS product will perform default format conversions.

Every DBMS column in a table has a name and a data type. The data type indicates to the DBMS how much physical storage to reserve for the column and the form in which the data are stored.

The DATE data type is handled in a different manner across different DBMSs. The ORACLE examples that follow were run on an HP-UX system, using SAS 6.12 TS020 and SAS Version 7. These demonstrate the use of dates using the default formats and data types, and how to change the defaults to suit your needs.

Using SAS Date and Time Values

The SAS System represents date, time and datetime values as numbers using the following rules:

- *A date is represented by the number of days between January 1, 1960 and that date.*
- *A time is represented as the number of seconds between midnight and that time of day.*
- *A datetime is represented by the number of seconds between midnight, January 1, 1960 and that datetime.*

The SAS System has a number of informats that read date and time values and convert them to SAS date and time values. The SAS System also has a number of formats that write date and time in a variety of ways.

SAS dates are valid back to A.D. 1582 and ahead to A.D. 20,000. Leap year, century, and fourth-century adjustments are handled correctly. However, leap seconds are ignored, and the SAS System does not adjust for daylight savings time.

You can create a SAS date constant or a SAS time constant by writing the date or time enclosed in single or double quotes, followed by a D (date), T (time), or DT (datetime) to indicate the type of value. Using the following patterns to create date and time constants:

`'ddmmm<yy>yy'D`

`"ddmmm<yy>yy"D`

represents a SAS date value as follows:

- `date='1jan1997'd;`
- `date='01jan96'd;`

`'hh:mm<:ss.s>'T`

`"hh:mm<:ss.s>"T`

represents a SAS time value as follows:

- `time='9:25't;`
- `time='9:25:19't;`

`'ddmmm<yy>yy hh:mm<:ss.s>'DT`

`"ddmmm<yy>yy hh:mm<:ss.s>"DT`

represents a SAS datetime value as follows:

- `dtime='18jan97:9:27:05'dt;`
- `if begin='01may97:9:30:00'dt then`
`end='31dec97:5:00:00'dt;`

Refer to the following example, on how SAS stores date, time and datetime values.

```
mydate = '31MAR1960'D;
```

```
/* Since SAS stores date values as the number of days between January 1, */  
/* 1960 and the date in question, we know that the variable mydate will be */  
/* numeric and have a value of 90. */
```

```
mytime = '00:15:00'T;
```

```
/* Since SAS stores time values as the number of seconds between midnight */  
/* and the time in question, we know that the variable mytime will be numeric */  
/* and have a value of 900. */
```

```

mydatetm = '02JAN1960:01:00:00'DT;

/* Since SAS stores datetime values as the number of seconds between */
/* midnight on January 1, 1960 and the datetime in question, we know that */
/* the variable mydatetm will be numeric and have a value of 90,000 */
/* (86,400 seconds in a day + 3,600 in one hour). */

put mydate = mytime= mydatetm= ;
run;

/* SAS LOG Result: MYDATE=90 MYTIME=900 MYDATETM=90000 */

/* We can use SAS formats to instruct SAS to display these values in a form */
/* that is more easily understood. Keep in mind that the format does not */
/* have any effect on the value stored in the data set, it only affects how the */
/* value is printed to output by the various SAS procedures and statements. */

data b;
  set a;
  format mydate mmddyy10.;
  format mytime time10.2;
  format mydatetm datetime20.2;
  put mydate= mytime= mydatetm= ;
run;

/* SAS LOG Result: MYDATE=03/31/1960
                  MYTIME=0:15:00.00
                  MYDATETM=02JAN60:01:00:00.00 */

/* It is important to note that there is nothing special about these time, date, */
/* or datetime variables. They are simply numeric variables to SAS. */
/* The value 900 has no intrinsic meaning absent some indicator to tell SAS */
/* what the 900 means. Oracle uses the column type attribute, SAS uses the */
/* format. Thus the variable mytime from the example above (mytime=900) */
/* can mean several different things depending upon which format is */
/* associated with it. If you print it with a time format, it means 12:15 A.M. */
/* If you print it with a date format, it means June 16, 1962. If you print it */
/* with a datetime format, it means 12:15 A.M. on January 1, 1960. This is */
/* shown in the example below and serves to illustrate the importance of */
/* format statements and type statements when loading SAS data into */
/* databases systems.

```

```

data c;
  set a;
  put mytime time8.;
  put mytime mmddy8.;
  put mytime datetime18.;
run;

/* SAS LOG Result: 0:15:00
                  06/19/62
                  01JAN60:00:15:00                */

```

For more information on SAS date and time values, refer to the manual, "SAS Language Reference, Version 6".

PROC DBLOAD

The DBLOAD Procedure converts the SAS date variable formats into ORACLE data types by default, as follows:

<u>SAS Variable Format</u>	<u>ORACLE Data Type</u>
DATETIMEw.d	DATE
DATEw.*	DATE
TIMEw.d**	NUMBER

*Includes all SAS date formats, such as MMDDYYw., JULIANw. and QTRw., etc.

**Included all SAS time formats, such as HHMMw.d, MMSSw.d, and HOURw.d.

Note: With PROC DBLOAD, ONLY SAS variables created using INFORMATs of DATE are compatible with the ORACLE DATE data type. SAS variables created using INFORMATs of DATETIME are NOT compatible and a FORMAT statement is required.

USING THE DEFAULT ORACLE DATA TYPES

The example below demonstrates how to load a SAS data set containing different SAS date formats into an ORACLE table using PROC DBLOAD with the default ORACLE data types. PROC DBLOAD passes the SAS format to ORACLE and determines from the format what default data type to use. Therefore, if the FORMAT statement below was not used then the default format would be numeric (in this case, BESTw., if the datetime, date and time formats were not used).

```

data dataset;
  input  @1 date1 datetime18. @20 date2 date7. @28 date3 time8.;
  format date1 datetime18. date2 date7. date3 time8.;
cards;
08JUL1996:16:24:43 08JUL96 16:24:43
12SEP1964:14:22:00 12SEP64 14:22:00
30OCT1996:08:02:11 30OCT96 08:02:11
;

proc dbload dbms=oracle data=dataset;
  user=usr_name;
  orapw=password;
  path="@oracle_alias";
  table=datetest;
  load;
run;

```

The following results are generated using the ORACLE utility, SQLPLUS.

```
SQL> select * from datetest;
```

DATE1	DATE2	DATE3
08-JUL-96	08-JUL-96	59083
12-SEP-64	12-SEP-64	51720
30-OCT-96	30-OCT-96	28931

```

***** DATE1 is defined as ORACLE data type date
***** DATE2 is defined as ORACLE data type date
***** DATE3 is defined as ORACLE data type number

```

```
SQL> describe datetest;
```

Name	Null?	Type
DATE1		DATE
DATE2		DATE
DATE3		NUMBER

OVERRIDING THE DEFAULT ORACLE DATA TYPES

*Using the TYPE statement with PROC DBLOAD provides the capability of changing the default ORACLE data types in the new table. The example below creates a new table and overrides the default ORACLE data type. In this case, the FORMAT statement was not used, therefore SAS used the default format BESTW. To create a table with the ORACLE date data type, you **must** use PROC DBLOAD with the TYPE statement as follows:*

```

data dataset;
  input  @1 date1 datetime18. @20 date2 date7. @28 date3 time8.;
cards;
08JUL1996:16:24:43 08JUL96 16:24:43
12SEP1964:14:22:00 12SEP64 14:22:00
30OCT1996:08:02:11 30OCT96 08:02:11
;

proc dbload dbms=oracle data=dataset;
  user=usr_name;
  orapw=password;
  path="@oracle_alias";
  table=datetest;
  type 1="DATE" 2="DATE";
  load;
run;

```

*In this example, notice that **FORMAT** statements are **NOT** used when creating the SAS dataset, hence the default SAS data types are numeric. If the **TYPE** statements are not used in **PROC DBLOAD**, then the **ORACLE** default data types are number. However, since the **TYPE** statements are used the default SAS variable formats were overridden.*

NOTE: *In this example, the SAS variable `date1` can not be converted to the **ORACLE DATE** data type because SAS **DATETIME** variables are not compatible with the **ORACLE DATE** data type. Therefore, this **TYPE** statement will generate the **ERROR** message:*

```

ERROR: Oracle INSERT Error: -1858. Rejected observation 1.
      Re: ORA-01858: a non-numeric character was found where a
          numeric was expected
ERROR: Insert failed for obs number 1.

```

*To avoid receiving this error message, use a **FORMAT** statement of **DATE** or **DATETIME** (which ever is appropriate) for the SAS variable `date1` when creating the SAS dataset.*

*The SAS variable `date2` can be converted to the **ORACLE DATE** data type because SAS **DATE** variables are compatible with the **ORACLE DATE** data type. Therefore, the **TYPE** statement is used to demonstrate how to override the SAS numeric format and convert it to an **ORACLE DATE** data type.*

*The **ORACLE SQLPLUS** utility will generate the **EXACT** same results as the previous example assuming that a **FORMAT** of `datetime18.` is used as recommended above.*

```
SQL> select * from datetest;
```

```
DATE1      DATE2      DATE3
-----
08-JUL-96  08-JUL-96    59083
12-SEP-64  12-SEP-64    51720
30-OCT-96  30-OCT-96    28931
```

```
***** DATE1 is defined as ORACLE data type date
***** DATE2 is defined as ORACLE data type date
***** DATE3 is defined as ORACLE data type number
```

PROC ACCESS

The ACCESS Procedure assigns the SAS variable formats to ORACLE data types by default, as follows:

<u>ORACLE Data Type</u>	<u>SAS Variable Format</u>
DATE	DATETIME16. (1582AD to 2000AD)

USING THE DEFAULT ORACLE DATA TYPES

The example below demonstrates how to retrieve ORACLE data into a SAS data set using PROC ACCESS with the default ORACLE data types. PROC ACCESS retrieves the ORACLE data type and determines what default data type to use. Therefore, if the FORMAT statement below were not used then the default format would be numeric (in this case, E12., if the datetime, date and time formats were not used).

```
libname alib '/Fully/Qualified/Path/for_access_descriptors';
libname vlib '/Fully/Qualified/Path/for_view_descriptors';
```

```
proc access dbms=oracle;
  create alib.datetest.access;
  user=usr_name;
  orapw=password;
  path="@oracle_alias";
  table=datetest;
  create vlib.datetest.view;
  select all;run;
```

PROC PRINT would generate the following report using the same datetest table data and data types from the previous PROC ACCESS example.

OBS	DATE1	DATE2	DATE3
1	08JUL96:16:24:43	08JUL96:00:00:00	59083
2	12SEP64:14:22:00	12SEP64:00:00:00	51720
3	30OCT96:08:02:11	30OCT96:00:00:00	28931

PROC CONTENTS would generate the following:

Variable	Type	Len	Pos	Format	Informat	Label
DATE1	Num	8	0	DATETIME16.	DATETIME16.	DATE1
DATE2	Num	8	8	DATETIME16.	DATETIME16.	DATE2
DATE3	Num	8	16	BEST22.	BEST22.	DATE3

OVERRIDING THE DEFAULT ORACLE DATA TYPES

If you want to override the default formats, you can modify the results by writing PROC ACCESS code similar to the following:

```
proc access dbms=oracle;
  create alib.datetest.access;
  user=usr_name;
  orapw=password;
  path="@oracle_alias";
  table=datetest;
  format date1 datetime7.date2 worddate10.date3 hour8.;
  create vlib.datetest.view;
  select all;
run;
```

PROC PRINT would generate the following report:

OBS	DATE1	DATE2	DATE3
1	08JUL96	July	16
2	12SEP64	September	14
3	30OCT96	October	8

PROC SQL Pass-Through

By default, the SQL Procedure Pass-Through Facility returns the following ORACLE data types as follows:

<u>ORACLE data type</u>	<u>SAS Variable Format</u>
DATE	DATETIME16.

OVERRIDING THE DEFAULT ORACLE DATA TYPES

If you want to override the default formats, you can modify the results by writing PROC SQL code similar to the following:

```
proc sql;
connect to oracle (user=usr_name orapw=password
                  path="@oracle_alias");
  create table sas.dataset as
    select date1 format=datetime7., date2 format=datetime13.,
           date3 format=hour8.
    from connection to oracle (select * from datetest);
disconnect from oracle; quit;
```

PROC PRINT would generate the following report:

OBS	DATE1	DATE2	DATE3
1	08JUL96	08JUL96:00:00	16
2	12SEP64	12SEP64:00:00	14
3	30OCT96	12SEP64:00:00	8

You can also use the `datepart()` and `timepart()` functions to store the value that is needed so the default format is overridden, as shown below:

```
proc sql;
connect to oracle (user=usr_name orapw=password
                  path="@oracle_alias");
  create table sas.dataset as
    select * from connection to oracle
    (select * from datetest);
disconnect from oracle;
quit;
data timetest;
  set sas.dataset;
  date1=timepart(date1);
  format date1 time8.;
run;
```

PROC PRINT would generate the following report:

OBS	DATE1	DATE2	DATE3
1	16:24:43	08JUL96:00:00:00	59083
2	14:22:00	12SEP64:00:00:00	51720
3	8:02:11	30OCT96:00:00:00	28931

Using SAS/ACCESS and SQL Views in a DATA STEP

Using SAS/ACCESS views descriptors and PROC SQL views to process date and times can be difficult to comprehend.

*Note: PROC SQL Views are READ ONLY (DBMS tables cannot be updated).
PROC ACCESS Views are READ and WRITE (DBMS tables can be updated).*

SAS/ACCESS Interface to ORACLE provides the capability to access ORACLE tables using view descriptors created using PROC ACCESS views or SQL views created using the PROC SQL Pass-Through Facility. The following samples were tested with both SAS/ACCESS view descriptors and PROC SQL Pass-Through views.

A WHERE statement on a data step that refers to an ACCESS view descriptor or SQL view can be used. However, if the WHERE clause conforms to the DBMS syntax then the WHERE clause will be passed directly to the DBMS for processing. For instance:

Consider the ACCESS view or SQL view MYVIEW that points to an ORACLE table and references the column date1, which is of data type DATE in ORACLE. Remember that the ORACLE data type DATE is equivalent to the SAS data type DATETIME.

```
data timetest;  
  set MYVIEW;  
  where date1 > '01JAN95'D;
```

Since you are attempting to use a SAS DATE literal on the WHERE clause and the ORACLE Server only understands DATETIME values, the query will not be CORRECT. The WHERE clause DATE literal '01JAN95'D is represented by the number of days between January 1, 1960 and that date. The ORACLE Server data type DATE is expecting a representation equivalent to the SAS variable type DATETIME. Therefore, the above query will return incorrect results.

If the WHERE clause does not conform to the DBMS syntax then SAS will not send this WHERE clause to the DBMS for processing and instead, will ask the DBMS to send over ALL the data and SAS will discard those rows that do not meet the criteria.

Now consider:

```
data timetest;  
  set MYVIEW;  
  where date1 > '01JAN95:00:00:00'DT;
```

Since both the literal on the WHERE clause and the column in ORACLE are of matching types and the WHERE clause conforms to the DBMS syntax, this WHERE clause will be passed along to ORACLE for processing with all corresponding performance benefits.

Hints: If you use SAS Data Step Functions or SAS specific operators on the WHERE clause, the WHERE clause will not be passed to the DBMS.

If the SQL view was defined with a WHERE clause, the WHERE clause on the data step will be ANDED with it.

LIBNAME Engine ACCESS Support

This new LIBNAME engine ACCESS support is only available with SAS Release 7.x. In V7 SAS, a user can assign a libname statement to a "view" or "schema" of a DBMS database and a connection can be made when the LIBNAME statement is executed. Every use of a two-level SAS name, e.g. MYLIB.MYTABLE, will cause the DBMS engine to dynamically read the DBMS MYTABLE metadata. Both reading DBMS data and writing DBMS data will be supported using either DATA step or any of the PROCs. This is possible because of Dynamic Libnames.

The LIBNAME engine support, by default, converts the SAS date variable formats into ORACLE data types as follows:

<u>SAS Variable Format</u>	<u>ORACLE Data Type</u>
DATETIMEw.d	DATE
DATEw.*	DATE
TIMEw.d**	DATE***

*Includes all SAS date formats, such as MMDDYYw., JULIANw. and QTRw., etc.

**Included all SAS time formats, such as HHMMw.d, MMSSw.d, and HOURw.d.

***Version 7 now defaults to ORACLE DATE data type instead of NUMBER for SAS Variable Formats of TIMEw.d.

The LIBNAME engine support, by default, returns the same ORACLE data types as the PROC ACCESS and PROC SQL Pass-Through Facility (refer to the PROC ACCESS and PROC SQL Pass-Through Facility date chart).

USING THE DEFAULT ORACLE DATA TYPES

The example below demonstrates how to load a SAS data set containing different SAS date formats into a ORACLE table , datetest, using the LIBNAME engine with the default ORACLE data types. The LIBNAME engine for ACCESS passes the SAS format to ORACLE and determines from the format what default data type to use. Therefore, if the FORMAT statement below was not used then the default format would be numeric (in this case, E12., if the datetime, date and time formats were not used).

```
libname ORA_CNCT ORACLE USER=user_name ORAPW=password;
data dataset;
  input @1 date1 datetime18. @20 date2 date7. @28 date3 time8.;
  format date1 datetime18. date2 date7. date3 time8.;
cards;
08JUL1996:16:24:43 08JUL96 16:24:43
12SEP1964:14:22:00 12SEP64 14:22:00
30OCT1996:08:02:11 30OCT96 08:02:11
;

data ORA_CNCT.datetest;
  set dataset;
run;
```

The following results are generated using the ORACLE SQLPLUS Utility.

```
SQL> select * from datetest;
```

```
DATE1      DATE2      DATE3
-----
08-JUL-96  08-JUL-96  01-SEP-97
12-SEP-64  12-SEP-64  01-SEP-97
30-OCT-96  30-OCT-96  01-SEP-97
```

```
***** DATE1 is defined as ORACLE data type date
***** DATE2 is defined as ORACLE data type date
***** DATE3 is defined as ORACLE data type date
```

```
SQL> describe datetest;
```

Name	Null?	Type
DATE1		DATE
DATE2		DATE
DATE3		DATE

OVERRIDING THE DEFAULT ORACLE DATA TYPES

If you want to modify the default formats and informats that are returned to SAS from ORACLE, you can modify the results similar to the following:

```
libname ORA_CNCT ORACLE USER=user_name ORAPW=password;

proc print data=ORA_CNCT.mytable;
  format date1 datetime7. date2 datetime13. date3 hour8.;
run;

-OR-

data mod_date;
  set ORA_CNCT.datetest;
  format date1 datetime7. date2 datetime13. date3 hour8.;
run;

proc print data=mod_date;
run;
```

Using SAS Dates with MACROS

Processing dates using the MACRO facility has caused some confusion and misunderstandings. Below are some examples with PROC ACCESS, PROC SQL Pass-Through Facility, PROC DBLOAD and the LIBNAME engine support, that should help clarify the process.

The following example demonstrates the use of two DBMS dates to restrict the query using the Proc SQL Pass-Through Facility.

```
/* If creating start and end macros within a datastep... */
call symput ("start", ' ' || put(strtdate, datetime16.) || "'");
call symput ("end", ' ' || put(enddate, datetime16.) || "'");

-OR-

/* If creating start and end macros with %let... */
%let dset=dataset;
%let start=01jun96:00:00:00;
%let start=%nrquote(')&start%nrquote(');
%let end=01jun97:00:00:00;
%let end=%nrquote(')&end%nrquote(');

%let incond=where date1 between
              to_date(&start, 'DDMONYY:HH24:MI:SS') and
              to_date(&end, 'DDMONYY:HH24:MI:SS');
```

```

libname sas '/Fully/Qualified/Path';

proc sql;
  connect to oracle( user=usr_id orapw=password
                    path="@oracle_alias");
  create table sas.&dset as
  select * from connection to oracle
  (select datel1 as d1,
        date2 as d2,
        date3 as d3
   from DATETEST
   &incond);
  disconnect from oracle;
quit;

```

PROC PRINT would generate the following report:

OBS	D1	D2	D3
1	08JUL1996:16:24:43	08JUL1996:00:00:00	01SEP1997:16:24:43
2	30OCT1996:08:02:11	30OCT1996:00:00:00	01SEP1997:08:02:11

The example below demonstrates the use of a macro to process dates using a WHERE clause in a SAS data step using an ACCESS or SQL view.

```

data _null_;
input datelim datetime20.;
call symput ('wherdate', put(datelim,datetime20.) );
cards;
01jan95:00:00:00
;
run;

%put &wherdate;

data timetest;
  set myview;      /* SQL VIEW of DATETEST table */
  where datel1 > "&wherdate"dt;
run;

```

This next example demonstrates the use of macros to retrieve dates from a DBMS table using the LIBNAME engine support, available at SAS V7.

```

libname ORA_CNCT ORACLE USER=user ORAPW=password;
libname sas '/Fully/Qualified/Path';

%let dset=sas.dataset;
%let start=19960601000000;
%let start=%nrquote(')&start%nrquote(');
%let end=19970601000000;
%let end=%nrquote(')&end%nrquote(');
%let incond="where date1 between
            to_date(&start, 'DDMONYY:HH24:MI:SS')and
            to_date(&end, 'DDMONYY:HH24:MI:SS')";

data &dset;
  set ORA_CNCT.datetest ( dbcondition=&incond );
  format date1 datetime7. date2 datetime7. date3 hour8.;
run;

proc print data= &dset;
run;

```

PROC PRINT would generate the following report:

OBS	DATE1	DATE2	DATE3
1	08JUL96	08JUL96	330208
2	30OCT96	30OCT96	330200

Valid SAS FORMATS for Overriding the Default DATE FORMATS

<u>ORACLE Data Type</u>	<u>Valid SAS Variable Formats</u>
-------------------------	-----------------------------------

DATE	DATETIMEw.d *ONLY*
------	--------------------

** Time Formats return valid time values which are not accurate times compared with original dates and times because ORACLE internally stores dates and times differently than what SAS requires. Date Formats do not return any value. Therefore, all Time and Date formats are not supported for overriding default DATE/TIME formats.

NOTE: Informats are accepted but are ignored using PROC SQL Pass-Through.

Conclusion

This paper discussed the various aspects of processing dates with SAS/ACCESS Interface to ORACLE on UNIX platforms. This discussion included date processing using the ACCESS Procedure, SQL Procedure Pass-Through Facility, the DBLOAD Procedure and the LIBNAME engine support available with SAS V7 for ORACLE. It also included specific issues involving date formats for ORACLE and using ORACLE dates with MACROS.

For detailed information on the SAS procedures, PROC ACCESS, PROC DBLOAD, PROC SQL PASS-THROUGH and the LIBNAME engine support for ACCESS, reference to the following manuals:

SAS/ACCESS Software for Relational Databases Reference, Version 6

SAS/ACCESS Software for Relational Databases Reference, Version 7

SAS/ACCESS Interface to ORACLE Usage and Reference, Version 6

Getting Started with SAS/ACCESS Software, Version 6

SAS Language Reference, Version 6

For detailed information on the DBMS data types or functions used with the examples in this report, please refer to your Database Administrator and your DBMS manuals.