

# Looking into Oracle with SAS: Designing a simple SAS Windows interface for an Oracle Database.

Allison Mercer, CTB/McGraw-Hill,  
Monterey, California

## ABSTRACT

Using the capabilities of Base SAS it is possible to build an interface that allows SAS users to update and maintain an Oracle database. Many database functions can be performed from the SAS platform. Additionally, by designing windows appropriate to the task at hand, support staff with no programming knowledge can create tables and update existing data structures.

For a SAS shop where most data processing is done with SAS Procedures, this interface is ideal. Programmers can use the powerful SAS statistical procedures to process their input data and then write the finished product directly into the database, where it can be easily extracted in the format desired by the end user.

This paper demonstrates the usage of the SAS interface, shows some of the easiest ways to “talk” to Oracle on the fly and describes how to design the attractive and “user friendly” SAS windows that form the interface.

## INTRODUCTION

What better place to store your data than in an Oracle database! A well-designed database offers a high level of security, access control and ease of retrieval of your data.

Compare clicking through a maze of folders (that have seemingly random names, to locate a file stored by someone else) with the ease of typing in a table name that consists of logical project elements. Once your data is in Oracle it is in fine shape. But how to put your SAS data into the database and how to extract data from Oracle ?

There are Oracle tools and other interfaces on the market that allow loading, manipulation and extraction of the data in many formats. But SAS programmers do not have to exit SAS to perform many functions of these interfaces. Once the underlying SAS/Access for Oracle connection has been set up by the systems administrator, programmers using Base SAS can utilize Oracle tables almost as if they were SAS data sets in a basic SAS library.

## COMMUNICATING WITH ORACLE FROM SAS

Connecting to the Oracle database is as simple as using a SAS library. In fact, it is the SAS LIBNAME statement that makes available the connection to the database.

On our system the LIBNAME statement to the prototype database STATSTAR looks like this:

```
libname ORA ORACLE PATH='STATSTAR.CTB' USER='amercer' PASS='amercer'  
Schema = 'amercer';
```

You will need an Oracle account and, most probably, you will need to contact your Database Administrator to get the path and schema information , as well as, the necessary permissions to add tables to the database.

Once the library statement is set up correctly for your system, the fact that the library “ORA” is an Oracle database is almost transparent to you in using SAS.

You can perform many of the operations you use to view and manipulate the data sets in a SAS data library.

## LOOKING AT THE DATABASE TABLES

To see the members of the library, aka, the tables in the database, use PROC CONTENTS.

```
PROC CONTENTS DATA=ORA._ALL_ NODS; RUN;
```

A listing looks similar to that of PROC CONTENTS run on a SAS library.

### 1. PROC CONTENTS run on the oracle database.

Notes from log:

```
NOTE: Libref ORA was successfully assigned as follows:
      Engine:          ORACLE
      Physical Name:   STATSTAR.CTB
```

```
NOTE: PROCEDURE CONTENTS used (Total process time):
      real time          0.50 seconds
      cpu time           0.11 seconds
```

Output listing:

```
The SAS System          10:19 Sunday, July 17, 2005    1

                                The CONTENTS Procedure

                                Directory

                                Libref          ORA
                                Engine          ORACLE
                                Physical Name    STATSTAR.CTB
                                Schema/User      amercer

                                #   Name          Member   DBMS
                                #   Name          Type     Member
                                #   Name          Type     Type

                                1   PETS01_CANINE_DOG_TOYS  DATA    TABLE
                                2   PETS01_CANINE_DOG_FOOD  DATA    TABLE
```

### 2. PROC CONTENTS run on a SAS library containing the same tables:

Notes from log:

```
NOTE: Libref CANINE was successfully assigned as follows:
      Engine:          V9
      Physical Name:   C:\PROJECTS\CANINE\saslib
```

```
NOTE: PROCEDURE CONTENTS used (Total process time):
      real time          0.63 seconds
      cpu time           0.02 seconds
```

PROC CONTENTS output listing:

```
The SAS System          10:19
Sunday, July 17, 2005    6

                                The CONTENTS Procedure

                                Directory
```

```

Libref          CANINE
Engine          V9
Physical Name   C:\PROJECTS\CANINE\saslib
File Name      C:\PROJECTS\CANINE\saslib

```

#	Name	Member Type	File Size	Last Modified
1	DOG_TOYS	DATA	17408	08Feb05:14:18:56
2	DOG_FOOD	DATA	17408	03Dec04:14:33:36

The PROC CONTENTS output from the database differs from the contents of SAS libraries in that it does not provide file size and date last modified.

To see the contents of a table in the database, again use PROC CONTENTS.

```
PROC CONTENTS DATA=ORA.A_DOGTOYS; RUN;
```

I have given examples of the output of PROC CONTENTS and PROC PRINT for database tables in the section on loading data below. PROC CONTENTS for database tables does not provide the number of observations.

### GETTING DATA FROM ORACLE INTO A SAS PROGRAM

Use the data step.

```

data a_dogtoys;
set ora.a_dogtoys;
run;

```

A\_DOGTOYS is now a data set in the work library. You can work with it just as you can with any other SAS dataset or save it to another SAS library.

### LOADING DATA INTO ORACLE FROM SAS

The SAS procedure PROC DBLOAD will load data into Oracle. To use DBLOAD you do not need a library statement. The database connection information is given in the procedure call.

```

PROC DBLOAD dbms=oracle data=dogtoys; /* SAS data set name */
Orapw="amercer";
User="amercer";
Path="STATSTAR.ctb";
Table=NEWTTOYS; /* Oracle table name */
Reset all;
Load;
run;

```

PROC DBLOAD is reliable, but it has one aggravating drawback—variable names must be 8 characters or fewer in length.

To circumvent this difficulty, use the data step. An example of the load of a data set that would have failed in DBLOAD follows.

```

data ORA.BIG_dogtoys;
set dogtoys; /* SAS data set with long variable names */
run;

```

When we examine the log and PROC CONTENTS output we see that the long variables names loaded correctly.

Notes from log:

NOTE: There were 1 observations read from the data set WORK.DOGTOYS.  
NOTE: The data set ORA.BIG\_DOGTOYS has 1 observations and 2 variables.  
NOTE: DATA statement used (Total process time):  
    real time            0.18 seconds  
    cpu time              0.05 seconds

Contents listing of new Oracle table:

```
                                  The SAS System                                  12:43 Sunday, July 17, 2005   7

                                                                                  The CONTENTS Procedure

          Data Set Name          ORA.BIG_DOGTOYS          Observations          .
          Member Type            DATA                    Variables             2
          Engine                  ORACLE                 Indexes               0
          Created                 .                   Observation Length   0
          Last Modified          .                   Deleted Observations  0
          Protection              .                   Compressed           NO
          Data Set Type          .                   Sorted               NO
          Label
          Data Representation     Default
          Encoding                Default
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat	Label
2	NUMBER_OF_TOYS	Num	8			NUMBER_OF_TOYS
1	TYPE_OF_TOY	Char	7		\$7.	TYPE_OF_TOY

PROC PRINT listing of ORA.BIG\_DOGTOYS:

```
                                  The SAS System                                  12:43 Sunday, July 17, 2005   10

                                                                                  Obs          TYPE_OF_          NUMBER_
                                                                                  Obs          TOY              OF_TOYS

                                                                                  1          FRISBEE          6
```

## DELETING TABLES FROM THE DATABASE

Use PROC DELETE.

```
proc delete data=ora.a_dogtoys; run;
```

Log notes:

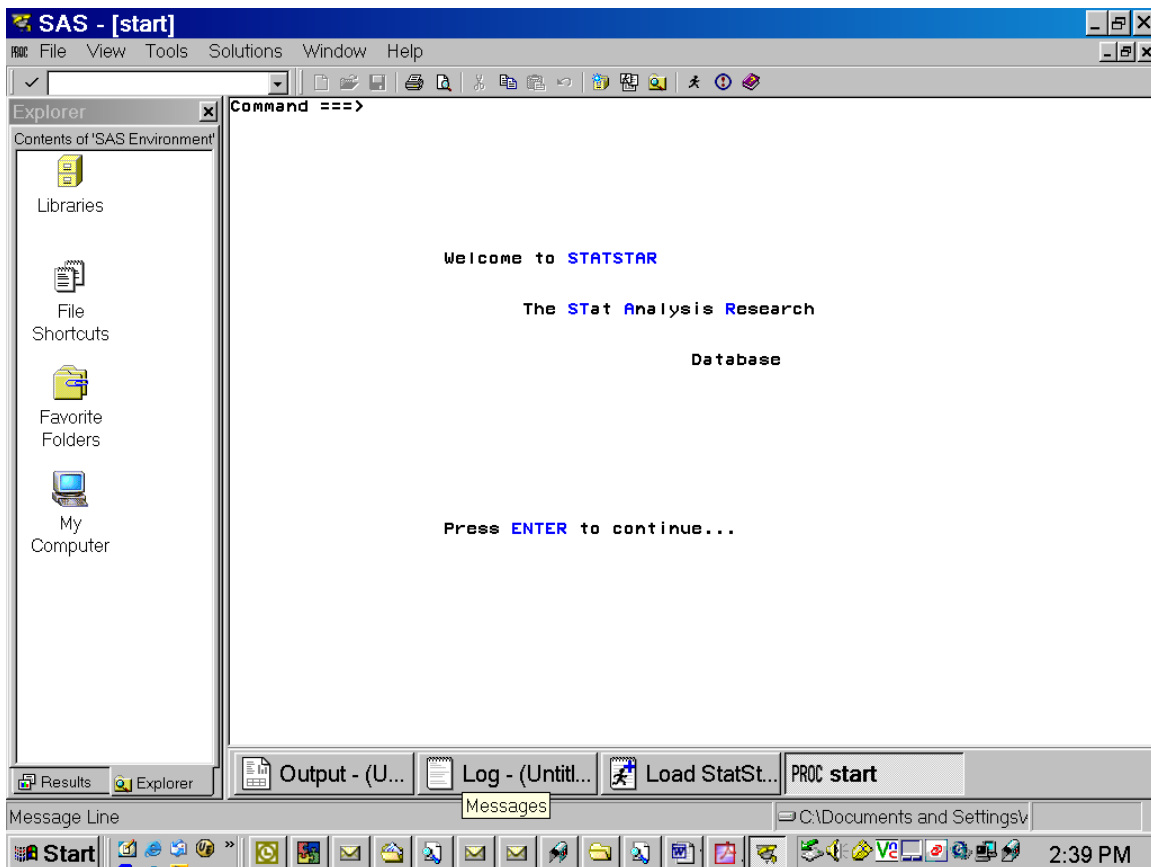
NOTE: Deleting ORA.A\_DOGTOYS (memtype=DATA).  
NOTE: PROCEDURE DELETE used (Total process time):  
    real time            1.74 seconds  
    cpu time              0.04 seconds

## USING SAS WINDOWS TO LOAD THE DATABASE

As we have seen, it is easy for a SAS programmer to write the code to connect to Oracle. If tables made within a program need to be put into the database, it is more efficient to add the code and load the tables directly from the program. However, in the case loading flat files from a network or a PC the use of an interface “cans” the procedure for the most common loads. The interface enforces naming conventions and makes the process one that easily can be performed by non-programmers. It also ensures that the variables that will become the primary keys for the database tables are correct.

The SAS WINDOW and %WINDOW statements are ideal for making an attractive and easy to use interface. Some examples of SAS windows I used in the interface to a prototype database, STATSTAR, and the code that produces them follow.

The Statstar Welcome window:



Code for Welcome window:

```
data _null_;
window start
  #9 @20 'Welcome to'
  #9 @31 'STATSTAR' COLOR=BLUE
  #12 @27 'The '
  #12 @31 'ST' COLOR=BLUE attr =BLINK
  #12 @33 'at'
  #12 @36 'A' COLOR=BLUE
```

```

#12 @37 'analysis '
#12 @45 'R' color=BLUE
#12 @46 'esearch'
#15 @42 'Parameter Database'
#25 @20 'Press'
#25 @26 'ENTER' COLOR=BLUE
#25 @32 'to continue...';

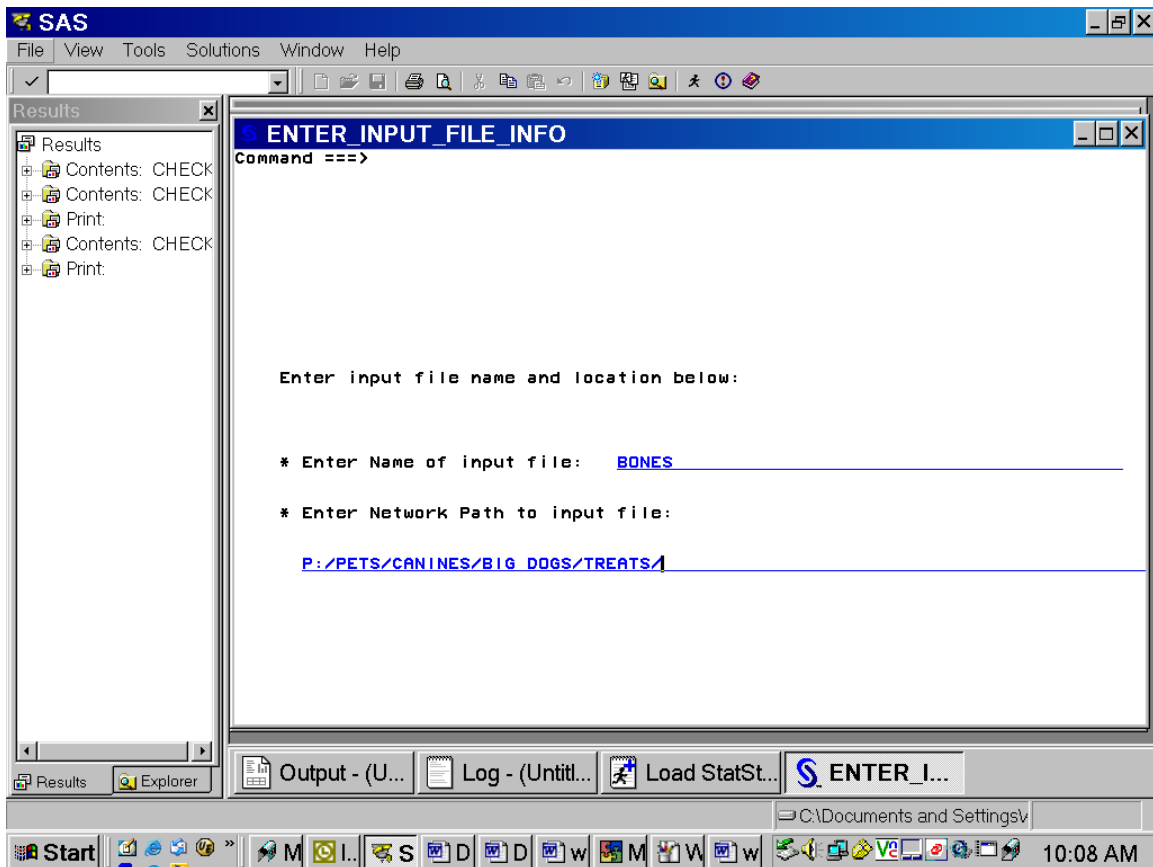
display start;
stop;
run;

```

The Welcome window is an example of a static window. It delivers unchanging information.

Other windows can accept information to pass to the underlying program. An example follows.

An information gathering window:



Code for information window:

```

%window ENTER_INPUT_FILE_INFO

#13 @5 "Enter input file name and location:"

```

```

#18 @5 "*" Enter Name of input file:"
    @35 inFile 45
        attr = underline required = yes color=blue

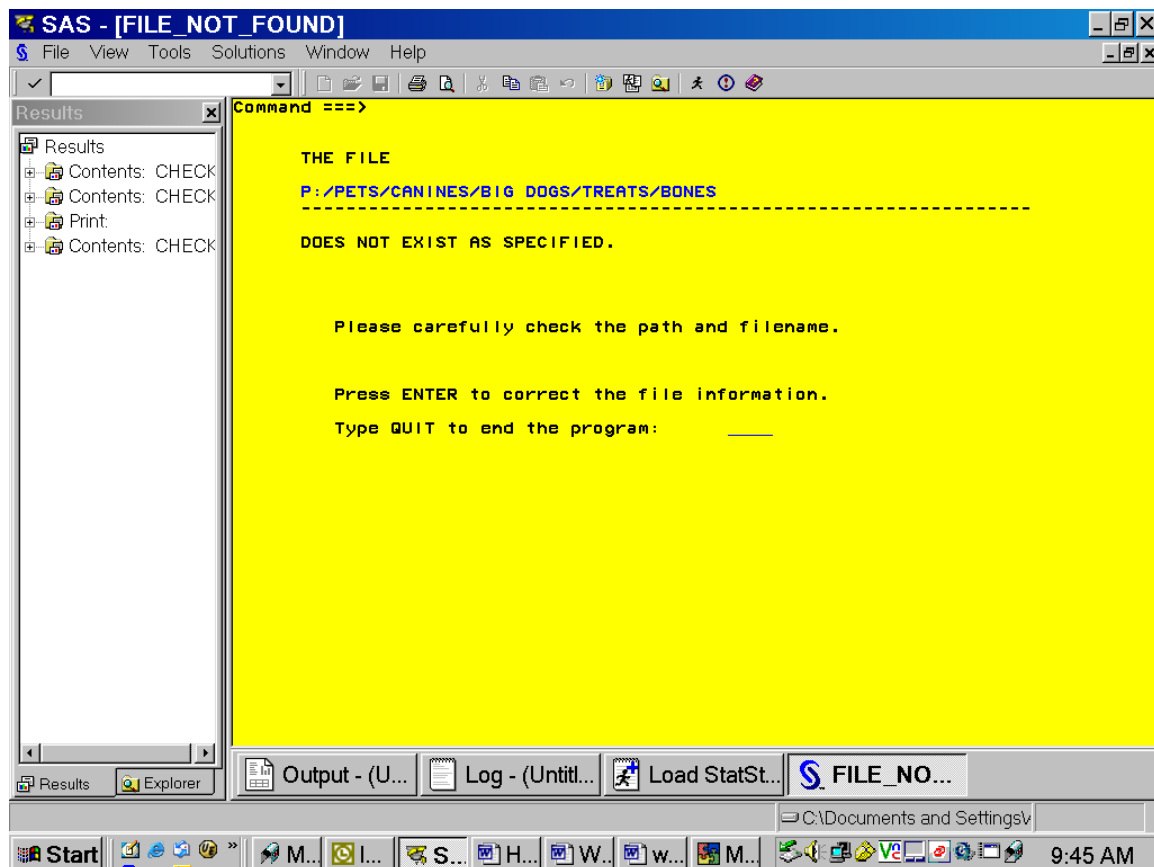
#21 @5 "*" Enter Network Path to input file:"
#24  @7 inPath 95
        attr = underline required = yes color=blue
;
%DISPLAY ENTER_INPUT_FILE_INFO;

```

This window accepts the information that you type in. It passes the name of the input file to the macro variable "inFile" and the path to the macro variable "inPath". The underlying program uses these macro variables just as it would macro variables made in any other way.

Other windows can show information from within the program.

A warning window:



Code for warning window:

```

%window FILE_NOT_FOUND color = YELLOW

/* dimensions of window */ icolumn = 1 irow = 1 columns = 90 rows = 30
/* Window content */

#3 @7 "THE FILE"

```

```

#5 @7 "&INPATH&INFILE" color=blue
#6 @7 "-----"
#8 @7 "DOES NOT EXIST AS SPECIFIED."
#13 @10 "Please carefully check the path and filename."
#17 @10 "Press ENTER to correct the file information."

#19 @10 "Type QUIT to end the program:" @45 abort 4
      attr = underline required = no color=blue
;

%DISPLAY FILE_NOT_FOUND BELL DELETE;

```

The FILE\_NOT\_FOUND window gathers information from the program and displays it. The program tested for the existence of the flat file "BONES". It passes back its result along with the information you typed in.

This window also accepts information, in this case, your decision whether to try to get the path and name correct or to quit the program. The program tests the macro variable "abort" that it receives through the FILE\_NOT\_FOUND window. If that variable does not contain the characters "QUIT" the program continues.

## CONCLUSION

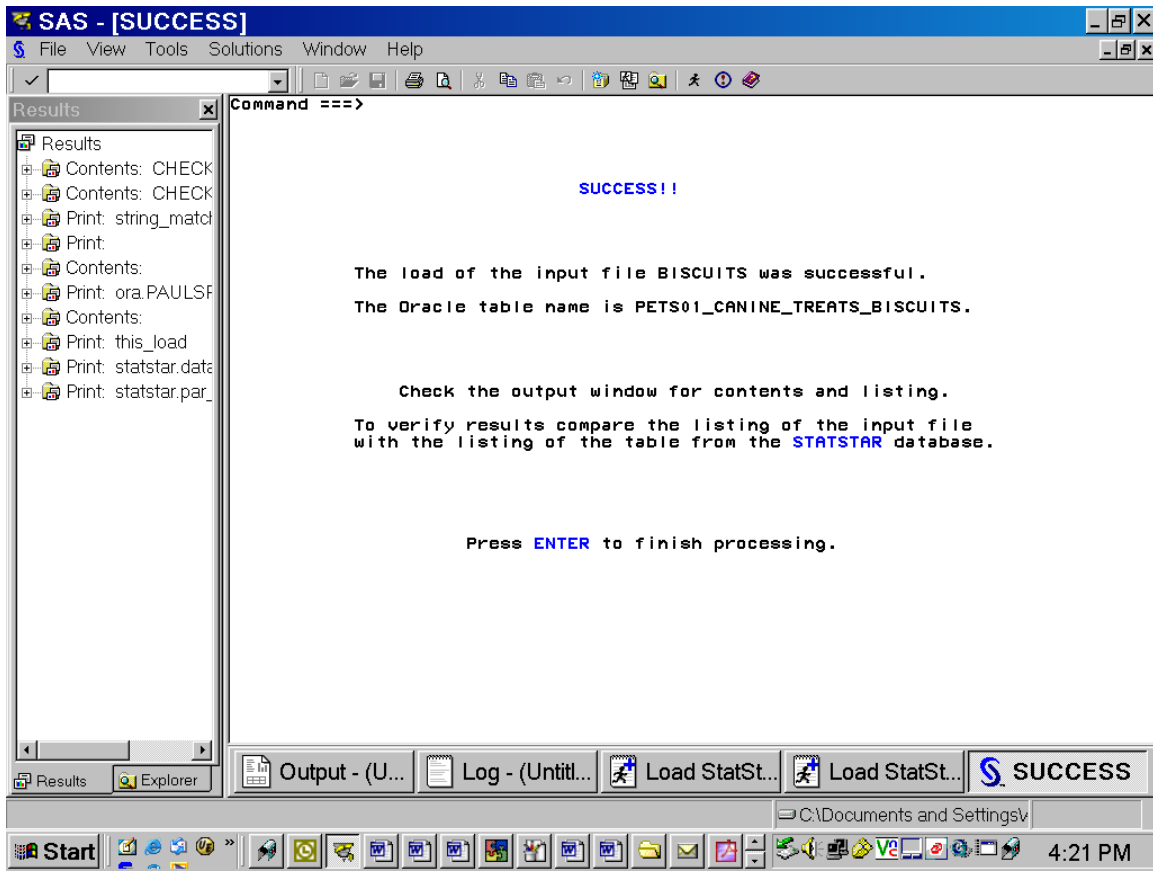
There are many doorways into an Oracle database. Investing in one of the many interfaces on the market, or hiring a database developer to design and construct an interface written a language such as 'PHP' is one option. For database developers and administrators such a front end to the database is a necessity.

But for the day-to-day entry of data into Oracle by many programmers, these options are expensive.

Any interface, in addition to the up front costs, has a learning curve which translates into time--time from every employee that needs to learn to use it. For companies already using SAS, it makes sense (and saves dollars) to utilize the convenient and easy relationship between SAS and Oracle.

By combining SAS code that accesses the Oracle database as if it were a SAS library and SAS windows that formalize and codify the procedures necessary to load the database we end up with one final window.

The STATSTAR success window:



SAS windows provide a clear view of Oracle to programmers and support staff alike.

## CONTACT INFORMATION

Allison Mercer  
CTB/McGraw-Hill  
20 Ryan Ranch Road  
Monterey, CA 93940

831-393-6709  
Allison\_mercer@ctb.com