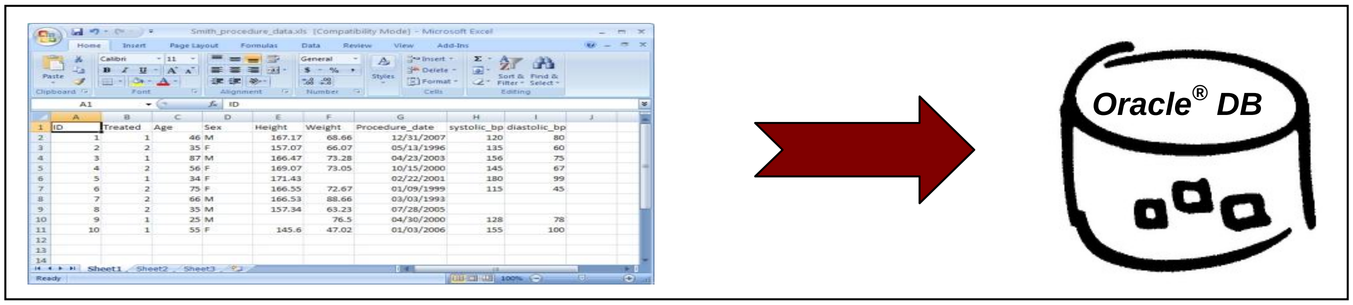
Paper 052-2010

Using SAS® Software's Metadata to Generate ETL Precursor Files in a Heterogeneous Database System

Alexander Pakalniskis, M.S., Cedars-Sinai Medical Center, Los Angeles, CA
Peixin Wang, M.S., Cedars-Sinai Medical Center, Los Angeles, CA
Nilesh Bajania, B.S., Cedars-Sinai Medical Center, Los Angeles, CA
Alein T. Chun, Ph.D., M.S.P.H., Cedars-Sinai Medical Center, Los Angeles, CA

Abstract

Created by the Resource and Outcomes Management (ROM) Department of the Cedars-Sinai Medical Center (CSMC), the Extract Transform Load System (ETLS) is an automated tool supporting the controlled movement of data from various input sources into Structured Query Language (SQL) environments (Figure 1). The tool, written in base SAS software and used in a Microsoft® Windows® environment, ensures ongoing information quality, reduces the time and effort required for data extraction, transformation, and loading, and provides audit report capabilities for project management. The output of this tool is designed to be used by database administrators (DBAs) in a linear or multi-layered heterogeneous database system.


Smith_procedure_data.xls [Compatibility Mode] - Microsoft Excel
Oracle® DB

	A	B	C	D	E	F	G	H	I
1	ID	Treated	Age	Sex	Height	Weight	Procedure_date	systolic_bp	diastolic_bp
2	1	1	46	M	167.17	68.66	12/31/2007	120	80
3	2	2	35	F	157.07	66.07	05/13/1996	135	60
4	3	1	87	M	166.47	73.28	04/23/2003	156	75
5	4	2	56	F	169.07	73.05	10/15/2000	145	67
6	5	1	34	F	171.43		02/22/2001	180	99
7	6	2	75	F	166.55	72.67	01/09/1999	115	45
8	7	2	66	M	166.53	88.66	03/03/1993		
9	8	2	35	M	157.34	63.23	07/28/2005		
10	9	1	25	M		76.5	04/30/2000	128	78
11	10	1	55	F	145.6	47.02	01/03/2006	155	100

Figure 1: *Input File to Oracle® Database*

Introduction

CSMC is located in Los Angeles, California and is one of the largest private academic hospitals in the Western United States. With nearly 1,000 beds, this medical facility's annual patient volume is more than 50,000 inpatient admissions and about 400,000 outpatient visits. Its work force comprises about 10,000 employees, 2,000 physicians on the voluntary staff and 200 in the clinical faculty.

ROM is the department which provides the institution with ad hoc, routine, and scheduled production clinical reporting, as well as support for data quality management issues. ROM is also responsible for the integrity of all data and reporting intended for recipients outside CSMC, such as governmental regulatory agencies or health consortiums.

Most ROM data reporting tasks consist of coding and executing SAS software embedded SQL queries to CSMC's data warehouse (Oracle® databases). The data that is provided to and used by the ROM staff also comes from a variety of other sources and formats, such as Microsoft Excel® files, Microsoft Access™ files, ".txt" files, SAS software data sets, etc. These data very often are required to be accessible for querying by other departments as well.

However, not all departments at CSMC have SAS software deployed as their preferred programming language. Some departments opt to use Crystal Reports®, Brio®, and a variety of other languages for their data manipulation needs. The common platform chosen by ROM for data access was an Oracle database, because it was possible for all these diverse languages to mine data from the data warehouse with embedded SQL queries (Figure 2).

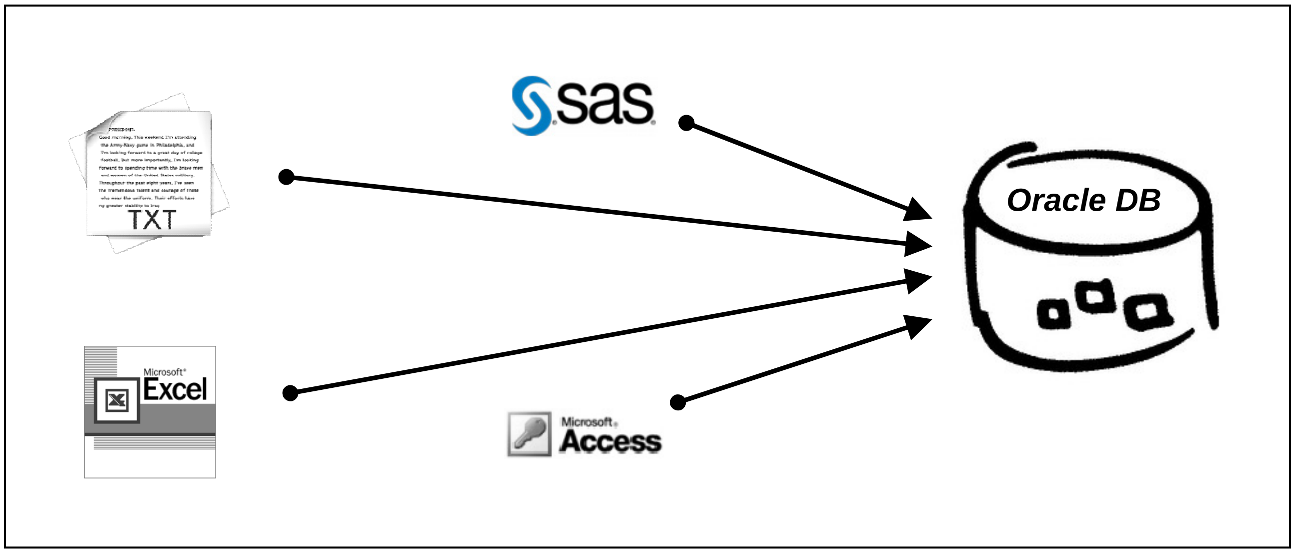


Figure 2: *Input Data Sources to Oracle Database*

Prior to the advent of ETLS, the method used to load input source data to the Oracle databases was to provide the DBA with information describing the characteristics of that data file. This ancillary information would include the location of the data file (network folder), the format of the file (Excel, ".txt", etc.), the number of records in the data file, the number of variables in the data file, the type of each variable in the data file (character or numeric), etc. If that incoming data set was relatively small, i.e., had about 5 variables and several hundred records, the DBA could manually create a table in the Oracle databases, load the contents of the data file to the table, and provide the user group with access to this table. However, when the user data file consisted of 170 variables and tens of thousands of records, the DBA's assignment became much more time consuming, resulting in a long wait period for these data to be loaded to the Oracle databases.

It was obvious that a much more efficient and error-free mechanism for importing data to an Oracle database was needed.

An analysis of the usability of a variety of tools was performed. For example, Toad® (originally this was an acronym standing for "Tool for Oracle Application Developers," but in 2004 "Toad" was registered as a trademark and was no longer considered an acronym) was found to be a very powerful and functional tool that could perform essential developmental and administrative tasks, such as data loading. However, it takes several hours to process large data files using Toad's load facility. And there are also numerous manual steps that were required of the DBA before Toad could start loading.

Oracle has a much more efficient tool for loading data into an Oracle database table. It is called SQL*Loader. Three files are required by SQL*Loader in order to load a file to an Oracle database: an executable script file (with an ".sql" suffix), an executable control file (with a ".ctl" suffix), and a data file (with a ".txt" suffix).

The script file prepares the Oracle database for the reception of incoming data by allocating space in the database structure for a table by naming that table, by identifying each variable name, its physical position on the incoming pipe-delimited ".txt" file, its type (character or numeric), and its length (in bytes), and by restricting access to the table (security). The control file identifies to the Oracle database which network folder contains the file to be loaded, instructs what to do in case the load is unsuccessful, describes the delimiter between variables, defines how the data is coming in on the input file, names the Oracle database table to load with this input data, dictates whether to append to the table or rebuild it, etc.

It was recognized that these three required files could be created by a program (using base SAS software) with almost no manual intervention (Figure 3). After all, both the script file and the control file used by SQL*Loader have a very rigid syntax, relatively easy to generate with a program. It was decided that the structure of the file to be loaded to an Oracle database by SQL*Loader would also follow a specific format, that of a pipe-delimited ".txt" file.

With these 3 files, a DBA can load 50,000 records of a ".txt" file in mere seconds.

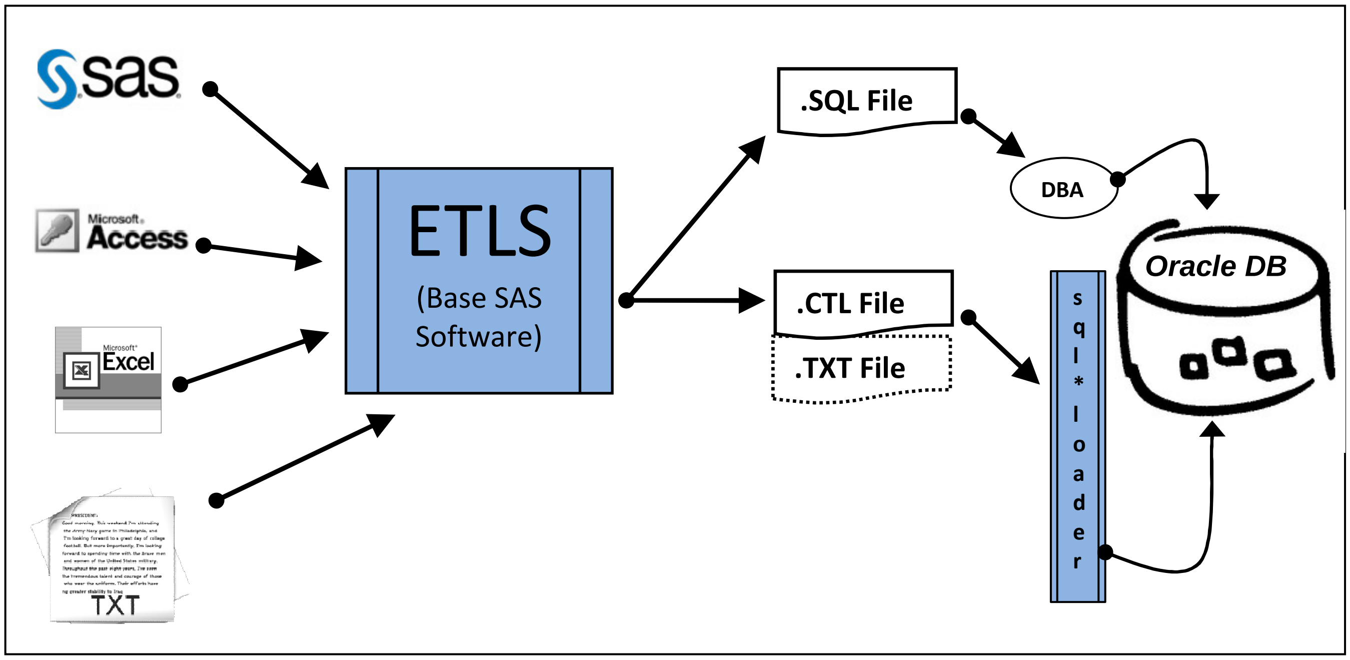


Figure 3: *Input Files to SAS software based ETLS to Oracle Database*

Using ETLS with Excel Files

The primary requirement in developing ETLS was to minimize manual intervention. In a perfect world it would be possible to create the ETLS so that it would be a simple “plug (point to the input file) and play (run the ETLS program)” episode. Unfortunately, input data is not always perfect and at some point in the process of loading to the Oracle database a manual inspection must take place. A major step in developing the ETLS process at CSMC was investigating ways to minimize this manual intervention.

The ETLS program for loading an input Excel file exemplifies this requirement for minimal manual intervention (see Step 1). Other than visual inspection of the user supplied input file, the process is completely automated and relies heavily on SAS software provided metadata.

Step 0: Initialize parameters

In the following initialization code &outdir is the network folder into which the three files (“.ctl”, “.sql”, and “.txt”) needed by SQL*Loader will be placed. &tblOwner, &tblSpace, &ora_ctl, and &ctlName are parameters whose values are determined by the DBA and provided to the individual executing the ETLS. For example, &ctlName identifies the name of the table to be created on the Oracle database and &ora_ctl is the network server into which the DBA will move the loadable “.txt” file (this is the staging library for loading to the Oracle database).

```
options       obs=max macrogen nosymbolgen mprint
              noxsync noxwait source2 nofmterr ls=256 ps=68;
%global numlong;
%let dirs = \\cshsmedaff\medaff\RO_Mgmt\shared;
%let outdir = &dirs\peixin;
%let infil = &outdir\9-15-09 3M 200903-200908.xls
%let tblOwner = ROM_OWNER;
%let tblSpace = ROM_DATA;
%let ora_ctl = /home/winnt/CSMC/bajanian/3m_drg;
%let ctlName = xls_data;
%let ident=Alex Pakalniskis;
%let outid=pakalniskisa@cshs.org;
filename outmail email  "&outid";
```

Step 1: Read and Cleanse Input File

PROC IMPORT is used to read a spreadsheet of an Excel file into a SAS software data set. Note that Excel spreadsheets have columns and rows, and an Excel cell represents the intersection of a column and a row.

- The “OUT =” argument identifies the library reference of the SAS software data set to be created.
- The "DATAFILE =" argument identifies the location of the file to be imported.
- The "DBMS" option identifies the input file as being an EXCEL2000 spreadsheet.
- The "REPLACE" option indicates that the SAS software data set named on the “DATAFILE =” argument, which this PROC IMPORT is to create, should be overwritten if it already exists.
- The "GETNAMES = YES" data source statement indicates that SAS software variable names should be generated from the contents of the input file's first row (the header row). If the cells of the header row contain special characters that are not valid in a SAS software variable name (such as a blank), the characters are converted to underscores.
- The "MIXED = YES" data source statement indicates that for those columns containing mixed data types (some cells with numeric data and others with character data), the import engine should assign a SAS software character type for the data found in that column and convert all numeric data values to character data values. (This feature is especially useful in the healthcare industry when dealing with diagnoses and procedures. A diagnosis code of "932" is designated as "Contusion of Upper Limb" while "932.00" is designated as "Contusion of Shoulder Region." The "MIXED = YES" data source statement guarantees that PROC IMPORT will represent the values in the resultant SAS software data set without removing trailing zeroes and/or decimal points. Changing a diagnosis from "932.00" to "932" in this step of the data transformation would be an error.)
- The "SHEET =" construct identifies a particular spreadsheet of the Excel file to be imported.

```
proc import   out = SAS_DATA_SET_FROM_EXCEL_1
              datafile = "&&infil"
              dbms = excel2000 replace;
              getnames = yes;
              mixed = yes;
              sheet = 'Sheet 1';
run;
```

The resultant SAS software data set of this PROC IMPORT event (SAS_DATA_SET_FROM_EXCEL_1) should be manually inspected at this point in the process. This is the only manual step of ETLS. If there are any unwanted or "ghost" variables, they can be eliminated with a DROP clause on the data statement. (These "ghost" variables are quite often named Fn, where n is an integer from 1 on upward. However, variables named F1, F2, ... may in fact be on the input Excel file and so as to preclude their deletion, this manual inspection task needs to occur.) If the input Excel file happened to be an empty Excel spreadsheet, PROC IMPORT would complete successfully and create a SAS software file with a single byte blank variable (F1).

```
data SAS_DATA_SET_FROM_EXCEL_2(drop=f1-f999);
 set SAS_DATA_SET_FROM_EXCEL_1;
run;
```

Step 2: Mine Needed Metadata from SAS Software Dictionary Table COLUMNS

The names, types (character or numeric), lengths (in bytes), and formats (for example "$9." or "Date9.") for all variables in SAS software file SAS_DATA_SET_FROM_EXCEL_2 need to be determined automatically. This metadata is found in SAS software DICTIONARY tables, which are special read-only SAS software data views containing information about SAS software data sets that are in use or available in the current SAS software session.

One of the ways to access these tables is by executing a PROC SQL query (within a SAS software program) using the DICTIONARY library reference (libref), which PROC SQL automatically assigns once invoked. So to get information from a DICTIONARY table, one needs only to specify DICTIONARY.table-name in the FROM clause of an SQL query.

Which DICTIONARY table should be used? How can one determine which DICTIONARY tables are available for use? The table DICTIONARY.DICTIONARIES contains information about all variables of all SAS software DICTIONARY tables.

```
proc sql noprint;
create table dictionary_dictionaries as
select *
  from dictionary.dictionaries;
quit;
```

Executing the previous SAS software embedded SQL query will identify which table to use in order to get the four pieces of information required by ETLS.

The output of the above SQL query (to SAS software data view DICTIONARY.DICTIONARIES), identifies DICTIONARY.COLUMNS as the DICTIONARY table containing the needed information. The following PROC SQL mines only the four pieces of information by ETLS from this dictionary by naming them in the SELECT clause. Additionally, the WHERE clause of the SQL query points to the library named WORK, the SAS software data sets of this library (memtype), and the data set being processed (memname).

```
proc sql noprint;
create table METADATA_OF_SAS_DATA_2 as
select upcase(name) as varName
      ,type         as varType
      ,length       as varLength
      ,format       as varFormat
  from dictionary.columns
 where libname = 'WORK'
   and memtype = 'DATA'
   and memname = 'SAS_DATA_SET_FROM_EXCEL_2';
quit;
```

Step 3: Ensure Variable Names Are Oracle Database Compatible

The objective of this portion of ETLS is to create data to be loaded into an Oracle database table. Since Oracle limits the length of column names to 30, it must be ensured that no variable name (column header) on the incoming file is longer than 30 characters.

Determining the length of the variable names on the incoming file is a simple task of checking the lengths of each varname of the SAS software data set METADATA_OF_SAS_DATA_2:

```
data SHOULD_WE_STOP(keep=sortsort varname);
 set METADATA_OF_SAS_DATA_2;
sortsort=0;
x=length(trim(left(varname)));
if  x > 30;
run;

proc sort data=SHOULD_WE_STOP nodupkey;
  by sortsort
     varname;
run;
```

If the SAS software data set SHOULD_WE_STOP were empty (i.e., there were no variable names on the incoming file longer than 30 characters), then in the following DATA _NULL_ step the macro variable &numlong would be created and assigned a value of zero. That's because the "END =" option on the SET statement creates a variable (lst) whose value is 0 until the DATA step starts to process its last observation. When processing of the last observation begins, then the value of variable lst changes to 1. Similarly, the automatic data set variable _N_ is initially set to 1. On the other hand, if SHOULD_WE_STOP were not empty, then the macro variable &numlong would be created and assigned a number equal to the number of variable names on the incoming file whose length is greater than 30.

```
data _null_;
if  lst
and _n_=1
    then call symput('numlong','0');
 set SHOULD_WE_STOP end=lst;
  by sortsort
     varname;
if  first.sortsort
    then jjj=0;
jjj+1;
if  last.sortsort;
call symput('numlong',trim(left(input(put(jjj,5.),$5.))));
run;
```

If there are variable names longer than 30 characters, they are identified in an email sent to the provider of the input file and the ETLS program is terminated. On the other hand, if all variables names are 30 characters or less in length, the ETLS program continues with the next step. The macro variable &numlong allows us to control the flow of ETLS:

```
%macro MWSUG;

       %if    &numlong > 0
              %then  %do;
                             /* this code generates email and ETLS is then stopped */
                     %end;
        %else
              %do;
                     /* this code represents the code for Step 4 through Step 9 */
              %end;

%mend MWSUG;

%MWSUG;
```

A workable variant of the SAS software code that generates email is:

```
data LIST_OF_LONG_VARIABLE_NAMES(keep=var1-var&numlong);
length var1-var&numlong $32;
retain var1-var&numlong i;
 set SHOULD_WE_STOP;
  by sortsort
     varname;
array varz(&numlong) $ var1-var&numlong;
if  first.sortsort
    then do;
            do i = 1 to &numlong;
               varz(i)=' ';
            end;
            i=0;
         end;
i+1;
varz(i)=varname;
if  last.sortsort;
run;

data _null_;
 set LIST_OF_LONG_VARIABLE_NAMES;
array arz(&numlong) $ var1-var&numlong;
file outmail
     to=("bajanian@cshs.org")
     cc=("pakalniskisa@cshs.org")
     subject="ETLS Rejection Notification"
     ;
put "Hello!";
put "          ";
put “&infil could not be processed.";
put "   ";
put "The following variable names are longer than 30 characters:";
put "    ";
do i = 1 to &numlong;
   put arz(i) $32.;
end;
put "    ";
put "Have a productive day!";
put "  ";
put "&ident";
put "Department of Resource and Outcomes Management";
put "Cedars-Sinai Medical Center" ;
put "Email: &outid";
run;
```

Step 4: Determine the Number of Variables on Input File from SAS Software Dictionary Table TABLES

Another source of useful metadata is the SAS software DICTIONARY table named TABLES. The mining of DICTIONARY.TABLES is limited to the library named WORK, to member types that are SAS software data sets, and to the member name of the metadata of the SAS software data set that was created from the user input file (see Step 2). Only one variable found on DICTIONARY.TABLES is of interest. This variable is named NOBS and represents the number of physical observations found in the metadata describing the input file.

The number of physical observations found in the metadata of the input file is the same as the number of variables in the input file. This is because each observation is a description of each variable in the input file.

In the SQL procedure of this step (see below) the SELECT INTO statement produces a SAS software macro variable &vCnt equal to the number of observations in the metadata file. (The -L in the PUT statement indicates that the value of NOBS should be left-aligned before being assigned to the macro variable.)

```
proc sql noprint;
select put(nobs, 8. -L) as vc
  into :vCnt
  from dictionary.tables
 where libname = 'WORK'
   and memtype = 'DATA'
   and memname = 'METADATA_OF_SAS_DATA_2';
quit;
```

Step 5: Determine Variable Names, Variable Types, and Variable Formats of Input File

Now that we know how many (&vCnt) variables are on the input file, three macro variables are created for each observation in the metadata of the input file. This is accomplished by the SELECT INTO statement of the SQL procedure.

The first observation in the metadata causes the creation of macro variables &varName1, &varType1, and &varFormat1, and the last observation in the metadata causes the creation of macro variables &varNameX, &varTypeX, and &varFormatX, where X is the value of &vCnt (=number of observations in the metadata file). The &varNameX macro variable holds the variable name, the &varTypeX macro variable is either “char” or “num”, and the &varFormatX macro variable is either a blank (for numeric data) or looks like “$7.” (for character data).

```
proc sql noprint;
select varName
      ,varType
      ,varFormat
  into :varName1 - :varName&vCnt
      ,:varType1 - :varType&vCnt
      ,:varFormat1 - :varFormat&vCnt
  from METADATA_OF_SAS_DATA_2;
quit;
```

Step 6: Determine the Maximum Length of Each Variable for Oracle Database Table Creation

Using the macro variables &varName1 through &varNameX (where X is the value of &vCnt), determine the length of each variable observed in the SAS software file equivalent of the Excel input file. The length of each variable needs to appear in the Oracle database table creating script that ETLS sends to the DBA. (This Data step and the following PROC SQL mine this information.)

```
data LENGTH_OF_VARIABLES(keep=lth1-lth&vCnt);
 set SAS_DATA_SET_FROM_EXCEL_2;
%do i = 1 %to &vCnt;
    lth&i = length(&&varName&i);
%end;
run;
```

Now that SAS software data set LENGTH_OF_VARIABLES contains all possible lengths for each variable, simply choose the maximum value for each variable and pin that value to macro variable &lth1 through &lthX, where X is the number of variables on the input file and ranges between 1 and &vCnt. Note that the macro variables created in this PROC SQL step are again the result of a SELECT INTO statement.

```
proc sql noprint;
%do i = 1 %to &vCnt;
    %global lth&i;
    select put(max(lth&i), 8. -L) as lx
      into :lth&i
       from LENGTH_OF_VARIABLES;
%end;
quit;
```

Step 7: Create a SAS Software Data Set from Which an External ".txt" Data File Will be Written

Create new SAS software data set (SAS_DATA_SET_FROM_EXCEL_3) from the SAS software data set of Step 1 (SAS_DATA_SET_FROM_EXCEL_2) with these modifications:

- All variables are of type character;
- The length of each variable is the maximum possible length the variable might have (as determined in the PROC SQL query of Step 6); and
- The variable names (column headers) are changed from whatever they were when the PROC IMPORT (Step 1) created them to newVar1, newVar2, …, newVarX, where X represents the number of variables in the input Excel file.

```
data SAS_DATA_SET_FROM_EXCEL_3(keep=newVar1-newVar&vCnt);
 set SAS_DATA_SET_FROM_EXCEL_2;
%do i = 1 %to &vCnt;
    length newVar&i  $ &&lth&i;
    %if  "&&varFormat&i" =  ""
         %then %do;
                  newVar&i = strip(&&varName&i);
               %end;
     %else
          %do;
              newVar&i = strip(put(&&varName&i, &&varFormat&i));
          %end;
%end;
run;
```

Step 8: Create SAS Software Data Sets Representing Oracle Database Script and Control Files

All the information mined from the user supplied input Excel file and SAS software Dictionary Tables (metadata of the user supplied input Excel file) accumulated to this point (Steps 1 through 7) is sufficient to create an executable script file and an executable control file with which the DBA will load the data provided by ETLS (in the form of a pipe-delimited ".txt" file that will be created in Step 9).

First, for ease of understanding the bounds of the DO loops in the following code, we create a new macro variable &vs, which is simply one less than &vCnt:

```
%let vs = %eval(&vCnt-1);
```

Prior to creating the external ".ctl" and ".sql" files, these files are written out as SAS software data sets. The first SAS software data set created is named tblScript and has one character variable (scrpt) of length 500. The second SAS software data set created is named ctl_file and has one character variable (ctl) of length 500. Both SAS software data sets are created with a PROC SQL statement.

```
%macro MWSUG2;

proc sql noprint;
create table tblScript (scrpt char(500));
insert into tblScript
       values("/*TABLE: &ctlname */")
       values("CREATE TABLE &tblOwner..&ctlname")
       values("(    ")
       values("     ")
       %do i = 1 %to &vs;
           values("     %left(&&varName&i)  varchar2(%left(&&lth&i)),")
       %end;
       values("     %left(&&varName&vCnt)  varchar2(%left(&&lth&vCnt))")
       values("     ) TABLESPACE &tblspace; ")
       values("    ")
       values("CREATE PUBLIC SYNONYM &ctlname FOR &tblOwner..&ctlname;")
       values("    ")
       values("GRANT SELECT ON &ctlname TO ROM_USER;");

create table ctl_file (ctl char(500));
insert into ctl_file
       values ("LOAD DATA")
       values ("INFILE '&ora_ctl/&ctlName..txt'")
       values ("BADFILE &ctlName..BAD   ")
       values ("DISCARDFILE &ctlName&..DSC")
       values ("APPEND")
       values ("INTO TABLE &ctlName")
       values ("FIELDS TERMINATED BY '|'");
quit;

data ctlxxx;
length ctl $ 500;
ctl = 'OPTIONALLY ENCLOSED BY '||"'"||'"'||"'";
output;
ctl = '(';
output;
run;

data ctl_file;
 set ctl_file
     ctlxxx;
run;

proc sql noprint;
drop table ctlxxx;
insert into ctl_file
       %do i = 1 %to &vs;
       values("     %left(&&varName&i) ,")
         %end;

         values("     %left(&&varName&vCnt)")
         ;
insert into ctl_file values(')');
quit;

%mend MWSUG2;

%MWSUG2;
```

Step 9: Write SAS Software Data Set Representations of Oracle Database Script and Control Files to External Files

```
filename outtxz "&outdir\create_XLS_script_file.sql";

data _null_;
file outtxz notitles ls=755;
 set tblscript;
put
    scrpt
       ;
run;

filename outtxt "&outdir\load_XLS_Control_file.ctl";

data _null_;
file outtxt notitles ls=755;
 set ctl_file;
put
       ctl
       ;
run;
```

Step 10: Create Pipe-Delimited ".txt" File

Now the pipe-delimited ".txt" file is created with the SAS software data set generated in Step 7.

Macros MWSUG and MWSUG2 have been previously defined and executed. In macro MWSUG an email was composed to list all variables within an incoming file whose variable names were longer than 30. We will now be defining macro SAS2TXT and then executing it. In macro MWSUG2 multiple variable names and variable lengths are referenced. Because this information is not known in advance, ETLS relies heavily on the use of macros.

The following macro, SAS2TXT, determines the number of spaces in the value of each variable.

```
%macro sas2txt;
```

If the variable doesn't have a value for a particular observation, the number of spaces is set to "-1" (negative one) in the data step. In the subsequent PROC SQL, new macro variables &spc1 through &spcX (where X is &vCnt, the number of variables in the Excel input file) are created. Each of these new macro variables equals the maximum number of blanks found in the value of a particular variable.

```
data sas2txt1;
length spc1-spc&vCnt 5;
retain spc1-spc&vCnt;
 set SAS_DATA_SET_FROM_EXCEL_3;
array s[&vCnt] spc1-spc&vCnt;
%do i = 1 %to &vCnt;
    spc&i = 1;
    do while ( scan(newVar&i, spc&i) ^= ' ');
       spc&i+1;
    end;
    spc&i+(-2);
%end;
run;

proc sql noprint;
%do i = 1 %to &vCnt;
    %global lth&i;
    select put(max(spc&i), 8. -l)  as lx
      into :spc&i
      from sas2txt1;
%end;
quit;
```

Having determined the maximum number of blanks in any variable, &vCnt new macro variables are created (Newlth1 through Newlth&vCnt). These new macro variables are equal to the “old” macro variables (Lth1 through Lth&vCnt) plus 3 times the number of blanks in each of the variables. This is done so that each blank in each variable can be replaced with the three characters “@#$”. (This three byte place holder (“@#$” will be converted back into a single blank immediately prior to writing the “.txt” file.)

```
%do i = 1 %to &vCnt;
    %let newlth&i = %eval(&&lth&i + 3*&&spc&i);
%end;

%do i = 1 %to &vCnt;
    %put &&newlth&i;
%end;
```

Change each space into triplet "@#$".

```
data sas2txt2;
 set sas2txt1;
%do i = 1 %to &vCnt;
    length txtvar&i $ &&newlth&i;
    txtvar&i = tranwrd(strip(newVar&i), ' ', '@#$');
    drop newVar&i spc&i;
%end;
run;
```

With the assurance that no variable has blanks in its value, a SAS software data set is created with one variable which is a concatenation of all variables, separated by “|”. This SAS software data set is the penultimate precursor to the pipe-delimited “.txt” file which the DBA will load to an Oracle database table.

```
data sas2txt3(keep=txtent);
set sas2txt2;
length txtent $ 10000;
txtent=%do i = 1 %to &vs;
           strip(txtvar&i)||'|'||
        %end;
strip(txtvar&vCnt)||'|';
run;
```

Change each triplet "@#$" (the three byte place holder of a blank) back into a blank.

```
data sas2txt4;
 set sas2txt3;
txtent = tranwrd(compress(txtent, ' '), '@#$', ' ');
run;
```

Write the pipe-delimited, SQL loadable file out to a network server for DBA to load.

```
filename outtx "&outdir\&ctlName..txt";

data _null_;
file outtx notitles ls=10000;
 set sas2txt4;
put
      txtent
      ;
run;

%mend SAS2TXT;

%SAS2TXT;
```

Examples of ETLS Usage

The University HealthSystem Consortium (UHC), headquartered in Oak Brook, IL, is an alliance of 107 academic medical centers (AMCs) and 233 of their affiliated hospitals, representing nearly 90% of the nation's nonprofit AMCs. The power of collaboration provided by UHC helps hospitals improve their clinical, operational, and financial performance. When UHC sends data to CSMC, it is in the form of a pipe-delimited ".txt" file and associated Excel metadata file. Clearly, the SQL*Loader required pipe-delimited ".txt" file requires no further transformation. The only two elements that require creation are the script and control files. In this instance ETLS uses the provided metadata file to produce these two files. UHC's mode of data delivery was the impetus for the development of ETLS.

The National Research Corporation (NRC), located in Lincoln, NE, provides a suite of patient satisfaction surveys in healthcare quality assessment and improvement through their Picker Institute's healthcare survey branch. The data flow between NRC and CSMC is bidirectional. CSMC creates an Excel file of patient demographic information for NRC's use in surveying these patients. Upon receiving survey results, NRC creates a SAS software data set of summarized survey results. Both data are processed by ETLS. The data transmitted to NRC uses ETLS as described in this paper for Excel files to create an Oracle database table in order to track the numerous iterations of survey populations. The data subsequently received by CSMC from NRC is processed by ETLS using the provided SAS software data set's automatically available metadata in the form of Dictionary Tables.

The U.S. Department of Health & Human Services provides a website (www.hospitalcompare.hhs.gov) which includes information that helps consumers compare the quality of all hospitals providing care for Medicare patients. This website allows the downloading of this information as an Access file. ETLS provides the facility to automatically load this file to an Oracle database table from which reports are subsequently created for hospital executive leadership.

One department here at CSMC produces Excel files with physician identification code, the clinical specialization with which the physician is associated, and other variable information needed by the department for internal use. While the physician identification code and clinical specialization have a fixed format on this file, and are also mandatory fields, all other fields can vary. For example, from one iteration to the next of this file, the variables may change in type (character or numeric), the number of variables might increase or decrease, etc. By using ETLS, manual intervention in determining what has changed from one iteration of that department's file to the next is no longer required at the Oracle database table loading step.

Conclusion

By performing automated extraction, transformation, verification, and load processes in place of manual tasks, ETLS saves both SAS software programmer and Oracle Database DBA time. The system also helps in avoiding errors natural to manual processes. ETLS has proven its value as a tool for ongoing program quality management and control. This system's service, available to all departments at our hospital, ensures the uniformity of data quality.

Acknowledgements

Thanks to Mirga Girnius, Ph.D. for editing and to Bruce Davidson, Ph.D. for reviewing this paper and providing their helpful remarks.

Recommended Reading

Carpenter, Art. Carpenter's Complete Guide to the SAS Macro Language, 2nd Edition. Cary, N.C.: SAS Press, 2004.
Lafler, Kirk. PROC SQL: Beyond the Basics Using SAS. Cary, N.C.: SAS Press, 2004.
Hand, David J., Heikki Mannila, and Padhraic Smyth. Principles of Data Mining. Cambridge, MA: MIT Press, 2001.
Huang, Kuan-Tsae, Lee, Yang W., and Wang, Richard Y. Quality Information and Knowledge. Upper Saddle River, NJ: Prentice-Hall, Inc., 1999.
Lee, Yang W., Pipino, Leo L., Funk, James D., and Wang, Richard Y. Journey to Data Quality. Cambridge, MA: MIT Press, 2006.

Contact Information

Your comments and questions are valued and encouraged.
Contact the authors at email address pakalniskisa@cshs.org.