Using SAS to get more out of Oracle Clinical

Jim Johnson, Covance Periapproval Services Inc., Radnor, PA

ABSTRACT
Oracle Clinical creates SAS data sets or SAS data views as part of its regular function. This paper will describe ways to get more out of Oracle Clinical than it “wants” to give you.

Oracle Clinical provides several types of data (Value Text, Full Value Text, DVG long value, DVG short value, DVG number, and Exception value text just to name a few). Each of these data types serves a specific purpose, and most will at some point become necessary for processing, reporting, and delivering the data to your user or client. Unfortunately, Oracle Clinical only allows a certain number of bytes for each record when generating the SAS data sets or views, and that width is not large enough to recover all the types of data available for data sets with a lot of variables. Only one ‘active’ extract is available at a time, so you need to know how to get the most out of it. Using SAS, you can use one extract get to all of these data types with little cooperation from the Oracle Clinical product.

DISCLAIMERS
Due to a limitation of space, this paper contains some of the details of how to process data from Oracle Clinical, but certainly not ALL the details.

There are always alternate methods of doing anything in SAS, some ways are better than others, some are more efficient than others, some are easier to understand than others. This paper is only able to present one way to approach these processes. You are encouraged to experiment with others.

The topics related in this paper are specific to:
- Oracle Clinical 3.1.1
- Oracle 7.3.3
- SAS 6.12

This paper employs an advanced technique hereafter referred to as “writing programs that write programs”. The technique will not be discussed in this paper. Please review the paper from the 2001 Pharmaceutical SAS Users Group Conference and the 2001 North East SAS Users Group Conference entitled “Programming Squared: Writing Programs that Write Programs” for details.

DEFINITIONS
Oracle Clinical stores all data as Full Value Text, which is always character variables of length 200. This way, any response to any question can be recorded. For instance, when asked to enter an age and the respondent says, “None of your business”, then that exact text can be entered into what is normally thought of as a numeric variable.

Oracle Clinical will provide various interpretations of the Full Value Text in your extract, as you request. Some of the choices are:
- Full Value Text – the character 200 value that was originally entered.
- Value Text – the converted version of the Full Value Text which takes on the expected attributes of the variable. For example, character 50, or numeric, or date.
- Exception Text – those Full Value Texts that do not convert into their assigned Value Text, because the length is too long, or the response is not numeric, or the response is in some other way not valid.
- DVG long value – The Discrete Value Group (often referred to as a code list) long value, which is the long version of the code conversion.
- DVG short value – the short version of the code conversion.
- DVG number – the code list numeric representation.

INTRODUCTION
This paper is written from a system which generates multiple SAS data libraries of SAS data sets. The libraries are:
- QCDATA: so called because it contains the data which quality control checks, or edit checks, are written against. This data is the Full Value Text representation of the data from Oracle Clinical.
- SASDATA: contains the converted values, or Value Text version of the data. This is the data the programmers and statisticians will use to analyze the data.
- EXCEPTION: this is the storage location for the invalid data that cannot be properly converted into the Value Text.
- COMMENTS: contains the investigator comments
- VALID: stores the Validation Status Flag.

Each of these libraries will contain the same data set names. Each like named data set in each library will have the same variables. Therefore, in all the data libraries above, each will have a data set named DEMOG, for instance. Each of the data sets named DEMOG will contain the variables SEX, RACE, AGE, HEIGHT, etc.

The advantage to this is that anyone can trace the value from the original QCDATA through to SASDATA and EXCEPTION. When a value in QCDATA will not properly convert to value appropriate to the SASDATA library, it will be placed in the EXCEPTION library. This way, if the SASDATA and EXCEPTION data sets are overlaid one upon the other the holes in the data in one data set will be filled by the presence of data in the other data set.

The disadvantage to this approach is that you cannot actually overlay the two data sets as described above.
because the variable attributes will almost certainly be very different from one another. Also, because the variables are named the same, you cannot conveniently MERGE the two data sets together so that the Value Text version of the data can be displayed alongside the Exception Text.

Other “forms” of the same data
Oracle Clinical allows the data entry operators to enter investigator comments from the case report form. For example: the question requests SEX, and the investigator has hand written a comment about the response, such as “transgender”. That comment is stored as an investigator comment. When extracted from Oracle Clinical it is given the variable name SEX, stored in a data set called DEMOG, and placed in a SAS library called COMMENTS.

Oracle Clinical also maintains Validation Status flags which indicate the current discrepancy status of any given piece of data. For the SEX question, this data will be extracted and given the variable name SEX, stored in a data set called DEMOG, and placed in a SAS library called VALID.

The investigator comments and validation status extracts are not going to be covered in this paper beyond that mentioned above.

WHAT YOU MIGHT WANT
Depending on how you use Oracle Clinical, you will have need for some, most, or all of the data representations offered by Oracle Clinical.
- If you process your edit checks outside Oracle Clinical, you will want access to the Full Value Text so your checks will run against exactly what was entered from the Case Report Form.
- Perhaps your client, be it another company, another department, or the FDA, would like to see the data exactly as it was entered. For that, you must have the Full Value Text.
- When it comes time to do the tables, listings, and graphs you will want to use the Value Text since programmers and statisticians would far rather process data using the numeric representation than the character 200 version.
- To produce easily read reports, you will need the DVG long or short value so you can report the number of Males and Females instead of the number of 1s and 2s.
- You will want to see the Exception Text so you know if any data that could not be converted needs further attention, or perhaps there are rules for processing items like partial dates. Partial dates, invalid dates, and unexpected responses will not appear in your Value Text.

There will probably be a need for all of these pieces of data during the processing of the project. Some of this you can do without, but most of it is necessary. Oracle Clinical allows you to request all of these forms of the data in your extract. Unfortunately, with earlier versions of Oracle Clinical and earlier versions of Oracle, there is a limit to how much data that can be extracted at one time.

Remember watching the Three Stooges and seeing the three of them trying to get through a doorway at the same time? It is the same for Oracle Clinical. The doorway is only so wide, and if you try to squeeze more through than will fit, you get nothing. If, for every variable you select the Full Value Text (which is always character 200), and the Value Text (which, for character variables, could be up to character 200), and the Exception Text (which is always character 200), the DVG long value (that can also reach character 200), along with all the Key Variables that you have no control over, you begin to create some very wide records and may exceed the extract width limitation.

The solution which quickly comes to mind is to create multiple extracts and use the one you need for your specific needs. Perhaps one extract that contains only the Full Value Text, and another that contains the Value Text. Unfortunately, Oracle Clinical supports only one active extract for each extract type (CURRENT, STABLE, TEST, etc.). Having two CURRENT extracts is not possible. One overwrites the other in Oracle Clinical.

Oracle Clinical stores only the Full Value Text. It generates all the other variations as the user requests them. If Oracle Clinical can operate with only one version of the text, then why can’t you? You can, and SAS can help you do it.

GETTING WHAT YOU WANT
Generate only one extract from Oracle Clinical. Request only the Full Value Text. Using the behind-the-scenes Oracle tables, create everything else needed for the processing and reporting of your data.

Depending upon the version of Oracle Clinical, and the features installed, there can be about 300 Oracle tables behind-the-scenes. They do not have names like DEMOG, AE, STUDYMED, or VITALS. Instead, there are tables with names like RESPONSES, RECEIVED_DCMs, QUESTIONS, QUESTION_GROUPS, and DISCRETE_VALUE_GROUPS. These 300 or so tables are interconnected through complex joins that give the illusion of the existence of CRF tables like DEMOG or VITALS.

Some of the more useful tables are:
- Clinical_studies – provides the clinical study number which is used in many of the other tables. Since the Oracle tables contain data for every study in a given instance of Oracle Clinical, this information is needed to help reduce the volume of data to be used.
- Dcm_questions – contains data about each individual question such as SAS name, data type, length, and precision.
- Discrete_values – gives the Discrete Value Group number, short value, and long value.
- Discrete_value_groups – provides general information about Discrete Value Groups.
- Questions – contains information about the questions such as SAS name, SAS label, status, and DVG.
- Received_dcms – has data about DCMs (Data Collection Module – basically, a Case Report Form
page) that have been received by the system. This table provides information about the DCM such as date received, pass one entered, pass two entered, and if it was received blank.

Using SAS/Access Software, you can get to the behind-the-scenes Oracle tables. A simple SAS program can pull these Oracle tables in as SAS views:

```sas
%macro ocview(table,dsn);
create view oc.&dsn as
select * from connection to db
(select * from &table);
%mend ocview;

libname oc "directory of your choice";
proc sql;
connect to oracle as db(user=xxxx orapw=xxxx path=xxxx);
%ocview(clinical_studies, clinstds);
%ocview(dcm_questions, dcm_q);
%ocview(discrete_values, dv);
.
.
.disconnect from db;
quit;
```

Run this program only once with all the Oracle tables you will need, and the views will always be available in the future just by executing the libname statement.

**PROGRAMMING TIPS**

There are a couple of programming techniques that will be used throughout this paper.

**Format modifier**

The ?? format modifier suppresses the printing of error messages and input lines when invalid data are read. It also prevents the automatic variable _ERROR_ from being set to 1. This means processing will continue even if invalid data are encountered. For example, the statement

```sas
x = input('abc',8.);
```

will cause an error because the character string 'abc' cannot be converted to numeric. However, the statement

```sas
x = input('abc',??8.);
```

will not cause an error. The value of x will be missing because 'abc' cannot be converted, and the program will continue. No warnings, errors, or messages of any kind will be produced as a result of this operation.

Further information about format modifiers can be found in The SAS Language Reference, Version 6, First Edition.

**Changing a variable's attributes without changing its name in one DATA step**

This technique is very simple. A data set contains the variable AGE stored as a character, but you desire the variable AGE to be numeric. Changing the variable's attributes in one data step is as easy as

```sas
data demog(rename=(num_age=age)
drop=age);
set demog;
num_age = input(age,??8.);
```

In this example, "on the way out" to the destination data set, SAS will first drop the original variable AGE, then rename the new variable NUM_AGE to AGE.

“If you use the RENAME= data set option in the same DATA step with either the DROP= or KEEP= data set option, the DROP= and KEEP= data set options are applied before the RENAME= data set option.” The SAS Language Reference, Version 6, First Edition, page 727.

**CLINICAL PLANNED EVENTS**

A listing of the Clinical Planned Events is not necessary for any processing discussed in this paper, but having a definitive list to provide those using the data will prove extremely helpful. Sometimes a long Clinical Planned Event name such as “Treatment period 2 cross-over washout” may be abbreviated. No one other than Oracle Clinical Start-up Team would know the value of the shortened version. Using a SAS view to the Oracle table clinical_planned_events will give the exact text for every Clinical Planned Event used in the study.

```sas
proc sql;
select visit_nu label='visit number'
,name label='Clinical Event'
from oc.cpe
where clinical = 1234
order by 1;
```

**DISCRETE VALUE GROUPS**

Obtaining the Discrete Value Groups is a key step. The DVGs are used to create all the formats necessary to convert, validate, and process the data.

By joining the behind-the-scenes Oracle tables discrete_values, discrete_value_groups, dcm_questions, and dcm_qs you can obtain all the DVGs and Qualifying Questions (which require DVGs) used in a study. The join will produce a table that provides the DVG number, DVG short value and the DVG long value. For example:

```
<table>
<thead>
<tr>
<th>#</th>
<th>short</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M</td>
<td>Male</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>Female</td>
</tr>
</tbody>
</table>
```

This data can be used to create a SAS data set to be used in PROC FORMAT with the CNTLIN= option to create the formats.

**SAS FORMATS**

These formats and informats are necessary to convert, validate, and report the data.
The Full Value Text will record the DVG short value (M and F).

- The character format will be used to convert the Full Value Text from M or F to 1 or 2 for use in calculations. This is the Value Text that will be stored in the permanent SAS data set.
- The numeric format will be used to convert the 1s and 2s to Male and Female for reporting.
- The character informat will be used to change the Male and Female back into Ms and Fs for the validation process.

CONVERSION TO VALUE TEXT

The concept of converting the Full Value Text to the Value Text is straightforward. The task is considerably more complex.

Half a dozen of the behind-the-scenes Oracle tables must be joined together to get a data set that describes the conversion that needs to take place. The data set will be similar to the one below. For future reference, this data set will be referred to in this paper as the meta-data data set.

<table>
<thead>
<tr>
<th>variable</th>
<th>length</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>$60</td>
<td>Name</td>
</tr>
<tr>
<td>type</td>
<td>$15</td>
<td>Data type</td>
</tr>
<tr>
<td>variable</td>
<td>$8</td>
<td>SAS name</td>
</tr>
<tr>
<td>length</td>
<td>8</td>
<td>length</td>
</tr>
<tr>
<td>decimal</td>
<td>8</td>
<td>decimal places</td>
</tr>
<tr>
<td>label</td>
<td>$40</td>
<td>SAS label</td>
</tr>
<tr>
<td>dvg</td>
<td>$30</td>
<td>DVG name</td>
</tr>
</tbody>
</table>

Once this data has been collected, it needs to be used to generate code to actually perform the conversion. A sample of the generated code is below. For convenience, only one variable is being processed. (A longer version of the generated conversion program is available in the appendix.) The line numbers are provided to trace the origin of the generated code.

```
proc format;
  value $sex /* character format */
    'M' = 1
    'F' = 2;
  value sex /* numeric format */
    1 = 'Male'
    2 = 'Female';
  invalute $sex /* character informat */
    'Male' = 'M'
    'Female' = 'F';

  The Full Value Text will record the DVG short value (M and F).
  • The character format will be used to convert the Full Value Text from M or F to 1 or 2 for use in calculations. This is the Value Text that will be stored in the permanent SAS data set.
  • The numeric format will be used to convert the 1s and 2s to Male and Female for reporting.
  • The character informat will be used to change the Male and Female back into Ms and Fs for the validation process.

  CONVERSION TO VALUE TEXT

  The concept of converting the Full Value Text to the Value Text is straightforward. The task is considerably more complex.

  Half a dozen of the behind-the-scenes Oracle tables must be joined together to get a data set that describes the conversion that needs to take place. The data set will be similar to the one below. For future reference, this data set will be referred to in this paper as the meta-data data set.

<table>
<thead>
<tr>
<th>variable</th>
<th>length</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>$60</td>
<td>Name</td>
</tr>
<tr>
<td>type</td>
<td>$15</td>
<td>Data type</td>
</tr>
<tr>
<td>variable</td>
<td>$8</td>
<td>SAS name</td>
</tr>
<tr>
<td>length</td>
<td>8</td>
<td>length</td>
</tr>
<tr>
<td>decimal</td>
<td>8</td>
<td>decimal places</td>
</tr>
<tr>
<td>label</td>
<td>$40</td>
<td>SAS label</td>
</tr>
<tr>
<td>dvg</td>
<td>$30</td>
<td>DVG name</td>
</tr>
</tbody>
</table>

  Once this data has been collected, it needs to be used to generate code to actually perform the conversion. A sample of the generated code is below. For convenience, only one variable is being processed. (A longer version of the generated conversion program is available in the appendix.) The line numbers are provided to trace the origin of the generated code.

  proc format;
    value $sex /* character format */
      'M' = 1
      'F' = 2;
    value sex /* numeric format */
      1 = 'Male'
      2 = 'Female';
    invalute $sex /* character informat */
      'Male' = 'M'
      'Female' = 'F';
  
  The Full Value Text will record the DVG short value (M and F).
  • The character format will be used to convert the Full Value Text from M or F to 1 or 2 for use in calculations. This is the Value Text that will be stored in the permanent SAS data set.
  • The numeric format will be used to convert the 1s and 2s to Male and Female for reporting.
  • The character informat will be used to change the Male and Female back into Ms and Fs for the validation process.

  A brief description of what this code is doing:
  • Lines 1 – 6 define the permanent data set SASDATA.VIT (which will contain the converted values), renames variables and drops variables.
  • Lines 7 – 12 define the permanent data set EXCEPT.VIT (which will contain the exception values), renames variables and drops variables.
  • Line 13 indicates the source data. In this case the libref points to the SAS view that looks directly into the Oracle Clinical data.
  • Lines 14 – 16 define temporary variables that will be used for the conversion. These variables will be renamed “on the way out” to the names expected in the final data sets.
  • Lines 17 – 18 does the actual conversion if a value exists.
  • Lines 19 – 20 checks to see if the converted value is missing and the source value is not missing. If so, the exception value is set to the source value.
  • Line 21 outputs the converted record (exception values are dropped “on the way out”).
  • Lines 22 – 24 if the exception text is not missing then the exception record is output (converted values are dropped “on the way out”).

  Each observation in the meta-data data set represents a different variable to be processed. Each variable needs to be included in both rename lists, both drop lists, the processing section, and the exception output section. These code groupings are shown below:

  ```sas
  data sasdata.VIT(rename=(
    c001 = HT1UNIT
  ));
  set qcd.VIT;
  attrib e001 label = "HT1UNIT Unit";
  attrib c001 format=8. length=8
  label = "HT1UNIT Unit";
  if HT1UNIT ^= "" then
    c001 = input(HT1UNIT ,??HT1UNIT_.);
  if c001 = . and HT1UNIT ^= "" then e001 = HT1UNIT ;
  output sasdata.VIT;
  if e001 ^= ""
    then output except.VIT;
  run;
  ```
```
These six parts are:
1. The naming of the first data set and the variables to be renamed.
2. The variables to be dropped from the first data set.
3. The naming of the second data set and the variables to be renamed.
4. The variables to be dropped from the second data set.
5. The body of the program where attributes are assigned and conversions are made.
6. A check of the exception variables to determine if an exception record needs to be output.

Each of these parts needs to be represented by a separate data set as the code is being generated.

```
data first /* section 1 */
dfirst /* section 2 */
second /* section 3 */
dsecond /* section 4 */
body /* section 5 */
last; /* section 6 */
set metadata end=last;
```

Using the ‘programs that write programs’ techniques, you can see that each section has a part that “has to be done first and only once”. Therefore you write code like this:

```
if _n_=1 then do;
  rec = "data sasdata.&dsn(rename=";  
  output first;
  rec = "drop=";  
  output dfirst;
  rec = "except.&dsn(rename=";  
  output second;
  rec = "set qcd.&dsn;";  
  output body;
  rec = "if";  
  output last;
end;
```

In the example given, two temporary variables are being created. One will contain the converted data (prefixed with a C), the other will contain the exception data (prefixed with an E). Processing of each will be discussed separately.

**EXCEPTION DATA**
The attributes of the exception data are the same as the Full Value Text, but the label is changed to match that of the Value Text.

```
rec = 'attrib e' || n || ' label="' || label || '"';
```

The temporary variable needs to be added to the rename list with its new name (line 8), and the original variable and the conversion variable need to be added to the drop list for the exception data set (line 11). Note the space in front of the ‘c’ to space the drop list properly.

```
output except.VIT;
```

Each exception variable needs to be tested to see if it is non-missing. One or more non-missing values indicate the exception record needs to be output.

```
rec = 'e' || n || ' ^=""';
```

**CONVERTED (VALUE TEXT) DATA**
The temporary variable needs to be added to the rename list with its new name (line 2), and the original variable and the exception variable need to be added to the drop list for the converted data set (line 5). Note the space in front of the ‘e’ to space the drop list properly.

```
output first;
rec = variable || ' e' || n;
```

The attributes of the converted data are specified by the meta-data data set. There are several different data types that need to be processed. The type could be specified as character (with or without a format/DVG), numeric, date, or time. Each needs to be processed separately.

**NUMERIC DATA**
The numeric attributes depend on the number of decimal places requested.

```
select (decimal);
when (0) round = '1 ';
when (1) round = '.1 ';
when (2) round = '.01 ';
when (3) round = '.001 ';
```

Likewise, the conversion also needs to account for the decimal places.
when (4) round = '.0001';
when (5) round = '.00001';
otherwise;
end;
rec = 'c' [ | n | | ' = round(input(' |
variable [ | ',?78.),'| |
round [ | ')');
output body;
rec = 'if c' [ | n | | ' = . and ' |
variable [ | '=""';
output body;
rec = 'then e' [ | n | | '="" |
variable [ | ';
output body;
end;

CHARACTER DATA
The attributes of character data depend on whether a
DVG is attached to the data. If there is a DVG, then the
stored data will be numeric.

if dvg ^= '' then do;
rec = 'attrib c' [ | n | | ' format = 8. length=8';
output body;
rec = 'label = "" | |
label [ | '=""';
output body;
rec = 'if ' [ | variable [ | "=" " then';
output body;
rec = 'c' [ | n | | ' = input(' |
variable [ | ',?78.); |
compress(dvg) [ | ')');
output body;
end;

If there is no DVG associated with the data, then
essentially only the length needs to be changed as
specified.

if dvg = '' then do;
rec = 'attrib c' [ | n | | |
' format=$ ' |
compress(length) [ | |
' . label = "" |
label [ | '=""';
output body;
rec = 'length c' [ | n | | ' = input(compress(' |
variable [ | ',?78.); |
compress(dvg) [ | ')');
output body;
rec = 'if c' [ | n | | ' = "" and ' |
variable [ | '=""';
output body;
rec = 'then e' [ | n | | '="" |
variable [ | ';
output body;
end;

DATE DATA
With date data it has proven beneficial to have the Full
Value Text carried along with the Value Text. This is
particularly useful when there are business rules for the
processing of partial dates. In the conversion process, if a
date is invalid (‘02/31/2000’ or ‘15/15/2000’) or the date is
partial (‘02/unk/2000’) the converted date will be blank.
The process below converts the Full Value Text to the
Value Text, and it also creates a new variable of a similar
name which contains the first several characters of the
Full Value Text, just in case the conversion fails.

rec = '_' [ | substr(variable,1,7);
output dsecond;
rec = 'attrib _' [ | substr(variable,1,7) |
' length=$20 |
' label="Original OC value for |
variable ' |
variable [ | ';
output body;
rec = '_' [ | substr(variable,1,7) |
' = input(compress(' |
variable [ | ',?78.); |
compress(dvg) [ | ')');
output body;
rec = 'then e' [ | n | | '="" |
variable [ | ';
output body;
end;

The regular date processing code looks like this:

rec = 'attrib c' [ | n | | ' format=date9. |
label = "" |
label [ | '=""';
output body;
rec = 'length c' [ | n | | ' = input(compress(' |
variable [ | ',?78.); |
compress(dvg) [ | ')');
output body;
rec = 'if c' [ | n | | ' = "" and ' |
variable [ | '=""';
output body;
rec = 'then e' [ | n | | '="" |
variable [ | ';
output body;
end;

When the last variable in the data set has been processed
(end=last option of the set statement), then the final
processing needs to occur. The rename statement group
and the data set options group each need to be closed,
the final converted observation, and exception
observation, if there is one, needs to be output.

if last then do;
rec = "output sasdata.&dsn";
output body;
rec = ')';
output first;
rec = ');';
output dfirst;
rec = '');';
output second;
rec = ');';
output dsecond;
rec = 'then output except.&dsn;';
output last;
rec = 'run;';
output last;
end;

All the pieces to convert the data from Full Value Text to
Value Text now exist and need to be assembled. They
will be assembled in a "flat file" that can be %included to
execute the generated code.

data _null_
file 'convert.pgm';
set first
dfirst
second
dsecond

VALIDATION
Using the same data sets, formats, and techniques above, a program can be generated to validate the conversion process. The validation step needs to recreate the Full Value Text and compare the results with the original data still stored in Oracle Clinical. The Full Value Text stored in Oracle Clinical is the Discrete Value Group short value, if a DVG is associated with the variable.

A sample of the generated code is below. For convenience, only one variable is being processed. (A longer version of the generated validation program is available in the appendix.) The line numbers are provided to trace the origin of the generated code. The code to convert back to Full Value Text needs to look something like this:

1 data new (drop=HT1UNIT ONE)
2(rename=(c1=HT1UNIT TWO));
3length c1 $200;
4format c1 $200.;
5set sasdata.VITS;
6c1 = left(put(HT1UNIT,z3.0));
7proc sort data=new;
8by docnum repeatsn qualifyv;
9proc sort data=except.VITS out=exc;
10by docnum repeatsn qualifyv;
11data both;
12update new exc;
13by docnum repeatsn qualifyv;
14proc sort data=qcd.VITS out=orig;
15by docnum repeatsn qualifyv;
16data orig;
17set orig; THREE
18if HT1UNIT ^= "" and
19input(HT1UNIT,??8.) ^=.
20then HT1UNIT =
21put(input(HT1UNIT,??8.),z3.0);

title3 ‘Date or time variables added to original data.’;
title4 ‘These will appear in the following PROC COMPARE as extra variables in the BOTH data set.’;
title5 ‘These values will NOT be considered during the PROC COMPARE since there is no match.’;
proc sql;
select name label = ‘Added Variable’
, label
from sashelp.vcolumn
where libname = ‘_SASDATA’
and memname = ‘VITS’
and substr(name,1,1) = ‘_’;
title3;
proc compare
   data = orig
   compare = both;
run; FOUR
============================================================================

A brief description of what this code is doing:
• Lines 1 – 5 define the temporary data set NEW, which will contain the Value Texts converted back to Full Value Texts, rename variables and drop variables.
• Lines 6 – 7 fix the length and format of the new variables that will contain the Full Value Text.
• Line 8 indicates the source data. In this case the libref points to the permanent SAS data set that contains the Value Texts created earlier.
• Line 9 does the conversion back to the character 200 Full Value Text. Here, a numeric variable is being changed to character and being zero filled to the precision defined by the Oracle Clinical design.
• Lines 10 – 11 sort the converted data by the minimum variables required to form a unique key for each record.
• Lines 12 – 13 sort the exception data by the same key.
• Lines 14 – 16 overlay the two data sets. The UPDATE function is being used to overlay the data sets since the two are complements, that is, where one has a blank field, the other may have data.
• Lines 17 – 18 sort the SAS view of the original Full Value Text data stored in Oracle Clinical.
• Lines 19 – 22 modify the appearance of the Oracle Clinical data by zero filling all the numeric variables. This is necessary because data is entered as it appears on the case report form. Sometimes leading zeros are used, sometimes they are not. Converting all numeric variables both in the Oracle Clinical view and the data set being compared to the same format will make the PROC COMPARE far more accurate.
• Lines following number 22 identify known added variables (the Full Value Text version of all dates) and then run the PROC COMPARE.

As before, the code can be broken into logical segments.
1. The naming of the data set and the variables to be dropped.
2. The variables to be renamed.
3. Setting of attributes and input data set (done when _n_ =1) and the actual conversion process. Lines 10 – 20 are added when the last observation in the meta-data data set has been processed.
4. Change the appearance of the Oracle Clinical numeric data to be zero filled for comparison. The lines after line 22 are added when the last observation in the meta-data data set has been processed.
The generation techniques are exactly the same as above. In the interest of space, this paper will not go into the same detail used above for each step. Attention will be paid to the conversion code.

NUMERIC DATA
Numeric conversion again needs to account for decimal places.

```
rec = 'c' || n || ' = left(put(', variable || ',z' || compress(length) || '.,' || compress(decimal) || ');';
```

CHARACTER DATA
If a DVG is associated with the variable, then the related format and informat need to be used to convert back to the Full Value Text. Using the example of SEX, Value Texts containing numeric 1s and 2s need to be converted first to the DVG long value using the numeric format. The data now contains 'Male' and 'Female.' These need to be changed to the DVG short value using the character informat. Now the data contains 'M' and 'F'.

```
c004 = input(put(sex,$sex.),$sex.);
```

The code to generate such a statement would look like:

```
rec = 'c' || n || ' = input(put(', variable || ',' || format || '),' || format || ');';
```

Character data with no associated DVG needs no fancy conversion.

```
rec = 'c' || n || ' = ' || variable || ';
```

DATE DATA
Date data conversion would be

```
rec = 'c' || n || ' = compress(put(', variable || ',yymmdd10.),'.-');'
```

Pay special attention to the use of single and double quotes in the statement above. The compress statement is necessary to remove unwanted characters. A successful conversion will produce a value like

```
1980-01-05
```

An unsuccessful conversion, or a missing value will produce a missing value. Thus removing the dashes will produce a date like the SAS view into Oracle Clinical produces, and removing the period (SAS missing) will make the value comparable to a missing value in Oracle Clinical.

CONCLUSION
The issues addressed in this paper may, or may not, be corrected in later versions of Oracle, Oracle Clinical, or SAS. The message from Oracle is that later versions of Oracle Clinical contain the same behind-the-scenes tables, variables, and relationships, with some additions. Therefore, the techniques used in this paper will not become obsolete with later versions of Oracle Clinical.

As with "programs that write programs", careful attention must be paid to single and double quoting. The right quotes need to be used at the right times, especially when dealing with macro variables, so that your generating statement produces the generated statement you need for the job.

Joining the behind-the-scenes tables is a complex task. Access to the technical manuals will help you understand their interrelationships. Being able to see the Oracle Clinical interface function – study set-up process to data entry – is also helpful to understand how the tables work and why they have been designed the way they are. A serious sense of adventurism, experimentation, and creativity will also add fuel to the process.

REFERENCES


TRADEMARKS
SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute, Inc. in the USA and other countries.

Oracle and Oracle Clinical are registered trademarks of Oracle Corporation, Redwood Shores, CA.

CONTACT INFORMATION
Jim Johnson
Principal Scientific Programmer
Covance Periapproval Services, Inc.
One Radnor Corporate Center, Suite 300
Radnor, PA 19087
jb.johnson@covance.com
This is a longer version of the generated conversion program. This version

- creates a new variable (_DOB) to carry a portion of the Full Value Text of the birth date variable (DOB)
- converts DOB to a SAS date
- converts SEX and RACE using the associated DVG
- converts a character string from $200 to $50
- converts character to numeric

```sas
data sasdata.DEMO(rename=(
c001 = DOB
c002 = SEX
c003 = RACE
c004 = ETHSPEC
c005 = AGED
)
drop=  
DOB  e001
SEX  e002
RACE  e003
ETHSPEC  e004
AGED  e005
)  
eexcept DEMO(rename=(
e001 = DOB
e002 = SEX
e003 = RACE
e004 = ETHSPEC
e005 = AGED  
)
drop=  
DOB  e001
SEX  e002
RACE  e003
ETHSPEC  e004
AGED  e005
)
set qcd.DEMO;
attrib e001 label = "Date of Birth";
attrib _DOB length=$20
label="Original OC value for variable DOB";
attrib c001 format=date9.
length c001 8;
if length(compress(DOB ,"-/")) = 8
then c001 = input(compress(DOB,"-"),??yymmdd8.);
else c001 = "";
if c001 = "" and DOB ^= "" then do;
e001 = DOB;
end;
attrib e002 label = "Sex";
attrib c002 format=8. length=8
label = "Sex";
if SEX ^= "" then
c002 = input(SEX ,??SEX.);
if c002 = . and SEX ^= "" then do;
e002 = SEX;
end;
attrib e003 label = "Ethnic Origin";
attrib c003 format=8. length=8
label = "Ethnic Origin";
```

Below shows the Proc Contents results of the non-key variables in data set SASDATA.DEMO.

<table>
<thead>
<tr>
<th>variable</th>
<th>length</th>
<th>format</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>_DOB</td>
<td>$20</td>
<td></td>
<td>Original OC value</td>
</tr>
<tr>
<td>for variable DOB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOB</td>
<td>8</td>
<td>DATE9.</td>
<td>Date of Birth</td>
</tr>
<tr>
<td>SEX</td>
<td>8</td>
<td>8.</td>
<td>Sex</td>
</tr>
<tr>
<td>RACE</td>
<td>8</td>
<td>8.</td>
<td>Ethnic Origin</td>
</tr>
<tr>
<td>ETHSPEC</td>
<td>$50</td>
<td>$50.</td>
<td>Ethnic specify</td>
</tr>
<tr>
<td>AGED</td>
<td>8</td>
<td>3.</td>
<td>Age Derived</td>
</tr>
</tbody>
</table>

Below shows the Proc Contents results of the non-key variables in data set EXCEPT.DEMO. Notice the absence of the variable _DOB.

<table>
<thead>
<tr>
<th>variable</th>
<th>length</th>
<th>format</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOB</td>
<td>$200</td>
<td></td>
<td>Date of Birth</td>
</tr>
<tr>
<td>SEX</td>
<td>$200</td>
<td></td>
<td>Sex</td>
</tr>
<tr>
<td>RACE</td>
<td>$200</td>
<td></td>
<td>Ethnic Origin</td>
</tr>
<tr>
<td>ETHSPEC</td>
<td>$200</td>
<td></td>
<td>Ethnic specify</td>
</tr>
<tr>
<td>AGED</td>
<td>$200</td>
<td></td>
<td>Age Derived</td>
</tr>
</tbody>
</table>
GENERATED VALIDATION PROGRAM
This is a longer version of the generated validation program. This version converts

- numeric to character with a zero filled format
- a SAS date to an Oracle Clinical format date
- character string from $50 to $200
- numeric values associated with DVGs to the DVG short values

```sas
data new (drop= AGED DOB ETHSPEC RACE SEX)
 rename=(
   c1=AGED
   c2=DOB
   c3=ETHSPEC
   c4=RACE
   c5=SEX
  )
 length c1 - c5 $200;
 format c1 - c5 $200.;
 set sasdata.DEMO;
 c1 = left(put(AGED ,z3.0));
 c2 = compress(put(DOB ,yymmdd10.),'.-');
 c3 = ETHSPEC ;
 c4 = input(put(RACE ,$RACE. ),$RACE. );
 c5 = input(put(SEX ,$SEX. ),$SEX. );
run;
```

```sas
proc sort data=new;
    by docnum repeatsn qualifyv;
run;
```

Below shows the Proc Contents results of the non-key variables in data set NEW.

<table>
<thead>
<tr>
<th>variable</th>
<th>length</th>
<th>format</th>
<th>label</th>
</tr>
</thead>
</table>
| _DOB     | $20    | $200.  | for variable DOB
| DOB      | $200   | $200.  | Date of Birth
| SEX      | $200   | $200.  | Sex
| RACE     | $200   | $200.  | Ethnic Origin
| ETHSPEC  | $200   | $200.  | Ethnic specify
| AGED     | $200   | $200.  | Age Derived
```

```sas
proc sql;
   select name label= 'Added Variable'
     ,label
     from sashelp.vcolumn
    where libname = '_SASDATA'
     and memname = 'DEMO'
     and substr(name,1,1) = '_'
;title3;
```