The National Occupational Respiratory Mortality System: Deployment of SAS/IntrNet® Application Dispatcher Queries Using SAS® Component Language (SCL) Macros

John M. Wood, National Institute for Occupational Safety and Health, Morgantown, WV

ABSTRACT

The National Institute for Occupational Safety and Health (NIOSH) has developed the interactive, Web-based, National Occupational Respiratory Mortality System (http://webappa.cdc.gov/ords/norms.html), a warehouse of national mortality data obtained annually since 1968 from the National Center for Health Statistics.

Long-term viability of the system required developing efficient algorithms for processing all possible variable combinations and outcomes while minimizing file size and maximizing the processing speed. File size was substantially reduced by representing cause-of-death outcomes as dummy variables (yes vs. no). Nested macro subroutines and macro variables were developed for reuse in similar processes. Examples include a subroutine for calculating numbers of deaths that also functions as the first step in calculating death rates; macro variables that control the processing of any combination of selected demographic criteria; and nested subroutines that process one or more disease selections. By using WHERE clauses with procedure statements, and by indexing prior to merging multiple data sets, we observed shorter processing time. Considering all enhancements, some queries with previous processing times of minutes now often execute in fewer than 10 seconds.

NORMS is a useful tool for implementation of a long-term NIOSH strategic goal of dissemination of occupational surveillance data for research purposes and disease prevention.

INTRODUCTION

The goal of the National Institute for Occupational Safety and Health (NIOSH), which is a component of the U.S. Centers for Disease Control (CDC), is to ensure the safety and health for all people in the workplace through research and prevention. One of many ways in which to facilitate the achievement of this goal is through timely dissemination of disease surveillance data. NIOSH has developed the National Occupational Respiratory Mortality System (NORMS) based on public-use, multiple cause-of-death (MCD) data files obtained annually since 1968 from the National Center for Health Statistics (NCHS) [1]. Usual industry and occupation codes are available for decedents from some states from 1985 to 1999. NCHS annually determines that certain quality criteria have been met by usual industry and occupation data from selected states. NIOSH verifies the results produced by NORMS against NCHS Control Total Tables to assure their accuracy.

NORMS includes a variety of demographic (age group, race, sex, Hispanic origin, region, state, county, usual industry and/or occupation, and time interval), clinical (underlying, contributing, or multiple cause-of-death), quantitative (death counts, death rates, years of potential life lost, and proportionate mortality ratios), tabular (sort order and data downloading), and graphical (charts and drill-down maps) query options. The "National Database" option includes those respiratory diseases or conditions that are clearly work-related. Other respiratory diseases (e.g., chronic obstructive pulmonary disease and lung cancer) that are not necessarily work-related can only be queried in the "Industry/Occupation Database." It is logistically impractical to create static files representing all possible outcomes, so an interactive, query-based Web application is an ideal conduit for data dissemination. Database queries are powered by the common gateway interface (CGI) component of SAS/IntrNet® and version 8.2 of the Application Dispatcher (AD). Both the NORMS data warehouse and the compiled SCL application code operate on a UNIX (Solaris) server located in Morgantown; the HTML interface is on a Web server in Atlanta, GA.

NORMS was originally developed with SAS/AF® software on a Windows Local Area Network (LAN) to serve as an in-house tool for generating reliable and reproducible statistics for a complex set of data. For public-use, Web-based applications, it is also important that results are produced quickly. Therefore, considerable effort has been spent on making the most recent version of NORMS both as fast and as statistically precise as possible. The objectives of this paper are: 1) to describe some of the methods used to organize the MCD data warehouse; 2) to explain how to pass parameters from the HTML form into the SCL code; 3) to show how to use macro code to process multiple-parameter queries, data downloads, and drill-down graphics; and 4) to review some effective procedures for debugging and deploying applications, and complying with Web-page accessibility guidelines.

METHODS

OBTAINING MCD DATA FILES

Batch programs were used to retrieve and transport a subset of the NCHS multiple cause-of-death data files from the CDC/Atlanta Data Center Mainframe System. The MCD system is based on a master list of alphanumeric codes published by the World Health Organization [2]. The NORMS subset includes any mention of a selected list of respiratory diseases or conditions that were clinically determined to be the underlying cause or a contributing cause of death [3]. MVS Batch processing was used to subset the available MCD data files, by year, 1968-2003. These 36 files were then converted to a binary format and transported (via File Transfer Protocol) from the CDC Mainframe to a UNIX platform The COPY procedure was used to reformat each binary file into a SAS® data set for use on the Solaris operating system.

ORGANIZING THE DATA WAREHOUSE

Collectively, the raw data files were too large to use with the interactive, Web-based application because each death record included codes for up to 20 conditions listed on the death certificate. However, the original demographic, clinical, and quantitative information for each individual death had to be retained in order to provide queries for any combination of those variables. A relational database structure was tried and rejected. Instead, a new file containing a dummy variable (yes vs. no) for each respiratory disease or condition included in NORMS [3] was created. A lot of time-intensive data manipulation was performed up front in order to make the Web-based queries run faster. An example of some of the methods used to manipulate the data is illustrated (for just a few of the respiratory diseases) in Figure 1. In addition to combining all diseases into one file, an indexed data set was created for each respiratory disease or condition included in NORMS. These data sets were permanently sorted by the year and by the original observation number in the raw data files.

INITIALIZING THE SELECTED PARAMETERS

The CGI component of AD is called the broker. In addition to this executable file (i.e., broker.exe), two other customizable files (i.e., appstart.sas and broker.cfg), must be installed on the application server [4].

The values of the query options selected on the HTML form are passed by the broker to the SCL application code when you activate the HTML Submit button. Here is a generalized example of how the statements in the HTML form should be written:

```
<form name="main" action="/path0/broker.exe"
            method="post" target="_blank">
<input  type="Submit"
     value="Submit National Database Query">
<input type=hidden name="_service"
     value="name_of_service">
<input type=hidden name="_program"
     value="path4.catalog02.norms.scl">
```

Note that the form tag includes an action statement for calling the SAS/IntrNet broker located at some designated UNIX path (here called path0) on the application server. The path to the application files containing the SAS code (here called path4) must be allocated in the appstart.sas file, while the name_of_service must be included in the broker.cfg file. The form also includes checkbox elements that allow you to select more than one respiratory disease or condition:

```
<fieldset><legend><label for="TopNational">
    Select Cause(s) of Death:
  <input type=checkbox name="disease"
     value="hyp">Hypersensitivity Pneumonitis
     <br>
     …additional input tag statements…
  <input type=checkbox name="disease"
     value="unsp">Unspecified Pneumoconiosis
  </label></fieldset><p> </p>
```

The elements of the HTML form are organized within fieldset, legend and label tags, and separated by paragraph tags, for compliance with Section 508 accessibility rules [5].

Figure 1. One of the techniques used to reduce the size of the SAS data sets used by the Web application.

```
libname master '/path1/';
libname norms  '/path2/';
%let fr=1968;
%let to=2003;

%macro YEARS1;
      %do yr=&fr %to &to;

  DATA mort&yr;   length mesothelioma
                  asbestosis silicosis $1;
       SET master.mort&yr;
           year=&yr;
           Y&yr=1;
           observation_num=_n_;

/* Initialize dummy variables */
           mesothelioma='N';
           asbestosis='N';
           silicosis='N';

/* Different codes apply to each ICD
   (only ICD-10 shown here)          */
  IF YEAR>=1999 THEN DO;
    array entity10(20) E1-E20;
    array ent10(20) $4 D1-D20;
     do E=1 TO 20 while (entity10(E) NE '');
      ent10(E) = substr(entity10(E), 3, 4);
       select (ent10(E));
        when ('J61 ') asbestosis='Y';
        when ('C450','C451','C452','C457',
              'C459') mesothelioma='Y';
        when ('J620','J628') silicosis='Y';
        otherwise; end;
     end;
   END;
   run;

 %end;
%mend YEARS1;
     %YEARS1;

%macro YEARS2;
     %do yr=&fr %to &to;
            mort&yr
      %end;
 %mend YEARS2;

/* Save all diseases in one data set */
data norms.all_diseases;
     set %YEARS2;
     run;

/* Save separate indexed data sets
   (only one example shown here)    */
data norms.asbestosis;
     length observation_num 5 asbestosis $1;
     set %YEARS2;
     if asbestosis='Y';
proc sort  data=norms.asbestosis;
     by year observation_num;
proc datasets  library=norms;
     modify asbestosis;
     index create
          id_num = (year observation_num);
     contents  data=asbestosis;
     run;
```

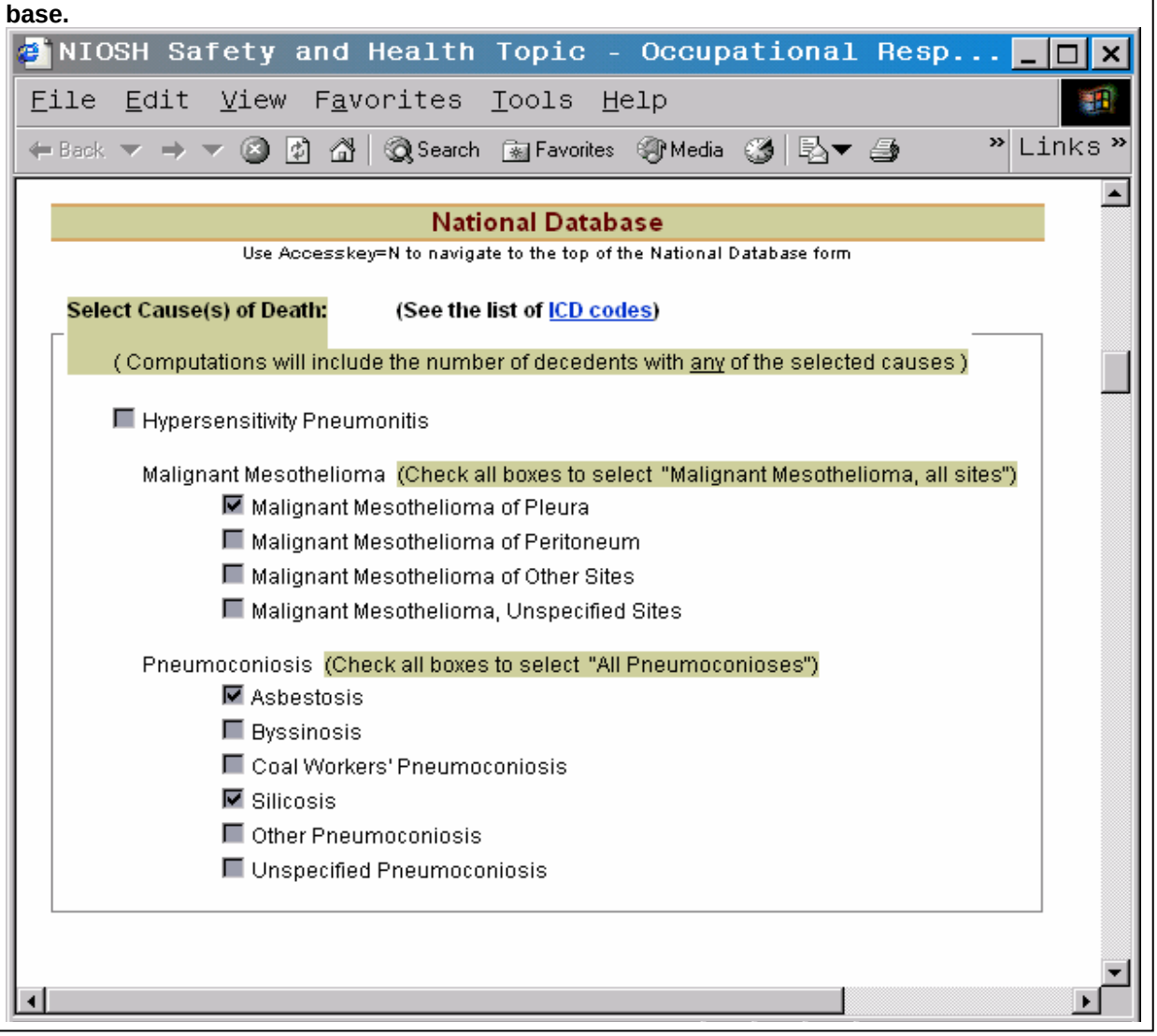
Each element in the HTML form includes a unique value. Passing the values of single-parameter query options into the SCL application code was simply a matter of initializing SCL variables for each parameter using the GETNITEMC function (Figure 2). The multiple-parameter disease checkboxes pass additional values through the CGI. Assuming that the three checked boxes shown in Figure 3 have been selected, the SAS broker receives a parameter that identifies the number of checked boxes (disease0=3), a parameter equal to the value of the first selected disease (disease=Mesothelioma), and a set of numbered parameters for each selected disease (disease1= Mesothelioma; disease2= Asbestosis; disease3= Silicosis). If only one disease is selected, then by default the broker will not receive any of the numbered parameters (i.e., disease0 – disease12). However, the HTML can be forced into always passing at least two numbered parameters (i.e., disease0 and disease1) by adding two blank values to the HTML form:

```
<input type=hidden name="disease" value="">
<input type=hidden name="disease" value="">
</form>
```

Figure 2. SCL functions used for initializing the values of some of the parameters passed by the SAS/IntrNet broker from the HTML form to norms.scl.

```
entry optional=cgivars 8;

INIT:
/* initialize SCL numeric variables */

  race=input(getnitemc(cgivars,"race"),3.);
  sex=input(getnitemc(cgivars,"sex"),3.);
  fr=input(getnitemc(cgivars,"fr"),4.);
  to=input(getnitemc(cgivars,"to"),4.);

/* initialize SCL character variables */

  state=getnitemc(cgivars,"state");
  service=getnitemc(cgivars,"_service");
  srvname=getnitemc(cgivars,"_srvname");
```

Figure 3. Multiple-parameter checkboxes in the HTML interface to the NORMS National Database.



Always passing at least two numbered parameters through the CGI made it easier to write the SCL code for processing either one or more than one disease parameter.

RETRIEVING SUMMARIZED DATA FOR ANY COMBINATION OF THE SELECTED PARAMETERS

Given the boxes checked in Figure 3, the broker will pass six numbered parameters, including two with null values (i.e., disease4 and disease5). However, as shown in Figure 4, the extra null parameters will be subtracted from the value of the SCL variable used to keep track of the number of selected diseases (nd). Macro subroutines (Figure 4, within the submit block) generate a set of SAS macro variables representing the selected diseases. Note that the SASdisease1-SASdisease&n variables generated in Part A are used in the Part B subroutine, and that Part B is nested within Part C. Parts D and E are likewise nested within Part F. Parts C and F do not immediately execute, but they are used in subsequent submit blocks (not shown) to calculate the number of decedents with any mention of the selected diseases. Since the value of state is never equal to **, that contingency in Part C is used solely to complete the WHERE clause that would otherwise result in a syntax error.

DEVELOPMENT AND DEPLOYMENT ISSUES

NORMS is a viable system that requires periodic maintenance and upgrades. There are occasionally coding errors in the existing application that have to be repaired, new data that are included in the warehouse, and enhancements or new ideas to work on. For various reasons, different security measures were taken on the development and deployment sides. Details and instructions for setting up private development interfaces and for Internet deployment are available from SAS technical support [4]. To debug the code described in Figure 4, a SAS/IntrNet socket service was defined in the broker.config file. Debugging was turned on for the socket service, and the following input tags were included in the HTML form:

```
<input type=hidden name="_service"
     value="socket">
<input type=hidden name="_debug" value="131">
<input type=hidden name="_program"
     value="path3.catalog01.norms.scl">
```

A separate, pooled service was established for application deployment. For security reasons, a separate SCL catalog was created with PROC BUILD using the NOSOURCE and NOEDIT options so that the compiled code could not be seen nor edited (Figure 5) using the BUILD procedure. No additional editing of the source code was necessary, since contingency statements referring to the separate services were built into the SCL code using the reserved AD _service variable (Figure 2). The debug option was turned off in the broker.config file using the DEBUGMASK option. A separate HTML file also was placed on the Internet server, with only the following input tags, to make use of the added security measures:

```
<input type=hidden name="_service"
     value="pooled">
<input type=hidden name="_program"
     value="path4.catalog02.norms.scl">
```

Figure 4. Continuation of Figure 2 code, completing the INIT section of norms.scl.

```
/* initialize more SCL variables */

  disease=getnitemc(cgivars,"disease");
  num_diseases=
   input(getnitemc(cgivars,"disease0"),3.);
  nd=num_diseases-2;

/* Use SCL variables nd and disease&n
   to create SAS macro variables and
   subroutines for later use */

submit continue;
  options mprint sgen;
  data _null_;


** Part A **;
  %macro multi_name;
   %do n=1 %to &nd;
      disease&&n="&&disease&n";
      call symput('SASdisease'||left(&&n),
           left(trim(disease&n)));
      %end;
    %mend multi_name;
   %multi_name;
** Part B **;
   %macro multi_set;
    %do n=1 %to &nd;
       &&SASdisease&n='Y' or
     %end;
    %mend multi_set;
** Part C **;
   %macro data_sum;
     proc summary data=norms.all_diseases
          (where=(%multi_set state='**'))
          nway;
      &&ClassDemographics;
      var Y&&fr - Y&&to;
      output  out=freqs  sum=;
        %mend data_sum;


** Part D **;
 %macro cond_inx_set;
  %do n=1 %to &nd;
    norms.&&disease&n (where =
   (&&disease&n='Y' and &fr<=year<=&to ))
  %end;
 %mend cond_inx_set;
** Part E **;
      %macro multi_conds;
       %do n=1 %to &nd;
          &&disease&n
        %end;
       %mend multi_conds;
** Part F **;
 %macro data_indexed;
   data freqs (keep =
        year %multi_conds Deaths);
   merge %cond_inx_set;
   by year observation_num;
%mend data_indexed;

    run;
  endsubmit;
RETURN;
```

Figure 5. SAS procedure for saving an SCL catalog that cannot be viewed or edited.

```
libname in   '/path3/';
libname out  '/path4/';

 proc build catalog = out.catalog02;
      merge catalog =  in.catalog01
                    nosource noedit;
      run;
```

If the debug tag is not removed, the DEBUGMASK option returns an error message to your Internet browser; either way, you will not be able to view the SAS logs.

RESULTS

NORMS includes nearly 17 million respiratory-related deaths out of the 75.3 million death certificates in the NCHS multiple cause-of-death database. Each of the 36 MCD SAS data sets (one for each data year, 1968-2003) has between 40 and 60 variables or fields. The total storage space for all 36 data sets is about 4.2 gigabytes (GB). The consolidated "National Database" warehouse that was created for use by the Web application (Figure 1) includes one 0.6GB SAS data set with 119 fields combining all 11 respiratory diseases, as well as 11 additional indexed SAS data sets (0.006GB or 6MB total) – one for each respiratory disease in the National Database (Figure 3). These 12 data sets are about 85% smaller than the original MCD data files that were obtained from NCHS. This process of data consolidation was an important first step in the overall methodology used to make the application queries faster and more efficient.

The difference in central processing unit (CPU) time for queries involving small versus large numbers of records is listed in Table 1. Comparison of the CPU times listed for Methods #1 and #2 shows how use of the WHERE clause can reduce processing time by as much as 70% by limiting the number of observations read. However, for multiple-disease queries involving data sets with a relatively small number of deaths, faster queries were achieved by initially merging indexed data sets and subsetting with a WHERE clause (Table 1, Method #3; Figure 4, Part F). Queries of diseases with relatively large numbers of deaths (e.g., chronic obstructive pulmonary disease and lung cancer) were processed faster by using the SUMMARY procedure (Table 1, Method #2; Figure 4, Part C).

Comparing the CPU times and debugging the SCL code was done by using the socket service previously described, since the SAS Log is written to the HTML file along with any output generated by the query. By including the MPRINT and SGEN global options in the submit block (Figure 4), macro code can be reviewed and debugged if necessary (Figure 6). The SYMBOLGEN statements describe what each macro variable resolved to. The MPRINT statements translate the macros into the actual code that was executed. The paired ampersand (&&) resolved to a single ampersand, thereby passing the n values of the SCL macro variables (disease1 – disease3) nested within the %DO loop. Contingency statements elsewhere in the SCL code (not shown) insert a CLASS statement (e.g., class sex race state;), or the blank value shown for &ClassDemographics in Figure 6, depending on the demographic options that were selected.

The HTML file that is returned to your Internet browser includes the SAS output generated by the query along with a submit button for downloading the tabular data to a spreadsheet file (Figure 7). The data that will populate the spreadsheet are saved in a temporary directory that is linked to the current Dispatcher session (Figure 8). Using an Institute-supplied macro called DS2CSV [6], a spreadsheet will be downloaded when you click on the download button, executing the SCL code in Figure 9.

Graphical results (GCHART and GMAP) can also be generated. NORMS outputs these as GIF files with mouse-over (charts and maps) and drill-down (maps only) capabilities (Figure 10). For example, the HTML option on the CHORO statement of the GMAP procedure (Figure 11) automatically fills in a hypertext URL for each state that contains the information needed to submit another query that will generate a county-level map.

Table 1. Central processing unit (CPU) times observed when accessing data sets with relatively small numbers (asbestosis and silicosis deaths, 1985–1999) versus data sets with relatively large numbers (chronic obstructive pulmonary disease and lung cancer deaths, 1985–1999). Methods #1 and #2 list the CPU times for processing a single SAS data set using the code described in Figure 4, Part C, except that Method #1 does not include the WHERE clause. Method #3 has the CPU times for processing two indexed SAS data sets using the code described in Figure 4, Part F.

Data Access Method	Relative Number of Deaths in the Data Sets	Number of Observations Read	CPU Time
#1: PROC SUMMARY without a WHERE clause	Small	1,653,867	26.82 seconds
	Large	1,653,867	27.44 seconds
#2: PROC SUMMARY with a WHERE clause	Small	28,996	7.93 seconds
	Large	879,674	17.40 seconds
#3: MERGE indexed data sets	Small	18,517	0.50 seconds
	Large	5,402,080	128.58 seconds

Figure 6. Determining if the macro code in Parts A–C of Figure 4 worked was accomplished by reviewing the SGEN and MPRINT statements generated in the SAS Log that was produced by a query of the three respiratory conditions selected in Figure 3.

```
SYMBOLGEN:  Macro variable ND resolves to 3
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable N resolves to 1
SYMBOLGEN:  Macro variable SASDISEASE1 resolves to mesothelioma
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable N resolves to 2
SYMBOLGEN:  Macro variable SASDISEASE2 resolves to asbestosis
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable N resolves to 3
SYMBOLGEN:  Macro variable SASDISEASE3 resolves to silicosis

MPRINT(MULTI_NAME):   disease1="mesothelioma";
MPRINT(MULTI_NAME):   call symput('SASdisease'||left(1), left(trim(disease1)));
MPRINT(MULTI_NAME):   disease2="asbestosis";
MPRINT(MULTI_NAME):   call symput('SASdisease'||left(2), left(trim(disease2)));
MPRINT(MULTI_NAME):   disease3="silicosis";
MPRINT(MULTI_NAME):   call symput('SASdisease'||left(3), left(trim(disease3)));

SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable CLASSDEMOGRAPHICS resolves to
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable FR resolves to 1999
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable TO resolves to 2003

MPRINT(DATA_SUM):   proc summary data=path3.all_diseases(where=(
MPRINT(MULTI_SET):  mesothelioma='Y' or asbestosis='Y' or silicosis='Y' or
MPRINT(DATA_SUM):   state='**')) nway;
MPRINT(DATA_SUM):   ;
MPRINT(DATA_SUM):   var Y1999 - Y2003;
MPRINT(DATA_SUM):   output out=freqs(drop=_type_ _freq_) sum=;
```

DISCUSSION

As the old sage Lao Tzu once said, “There are many paths to enlightenment…“ To those existential application developers whom have found their own way, I say congratulations! To those of you who may still be debugging, may you find some helpful suggestions among the preceding examples?

In my experience, a fast, efficient Web application is dependent first and foremost on the composition of the data warehouse. I tried to consolidate the raw data into the fewest number of records (rows) and fields (columns). For data sets with large record sizes, I found that a WHERE clause placed within a procedure statement substantially reduced application processing time. Data indexing in combination with a WHERE clause reduced CPU times even more for small numbers of records. Using these techniques, CPU times for the query results shown in Figures 7 and 10 were less than 10 seconds. I used compiled SCL code mainly because it was more secure than normal SAS program code. However, compiled code may also execute faster than interpreted code and uses less computer memory.

Macro subroutines can be designed to be reused for a variety of query options. In NORMS, the same macro code that is used to summarize the total number of deaths was also used in other calculations, such as in the calculation of death rates. Other subroutines were used repeatedly to subset any combination of demographic data, to create the titles and notes generated on the output pages, etc. The generalized code in Figure 11 only works for a query of the number of deaths for the entire U.S. However, I was able to develop drill-downs for any combination of the demographic, clinical, and quantitative options by adding additional contingency statements and substituting macro variable values for all of the fields in the SAS DATA-step and procedure code.

Up to this point, I have only mentioned Section 508 compliance [5] with respect to the fieldset, legend, and label tags that were used to group the HTML form elements. Anyone who has ever developed a government Web site should be familiar with this. To more easily comply with Section 508 Accessibility rules, NORMS does not rely on any JavaScript for data validation. All data validation takes place on the server side in the SAS application code. The SAS Output Delivery System (ODS) Accessibility Tagset [7] was used for formatting tabular output. I just recently learned that SAS has added an Accessibility macro for use with Version 8.2 that will associate descriptive text with ODS graphical output [8]; and that compliance is even easier using Version 9.1.3 of SAS [9]. Many government Web sites either do not seem to take these rules seriously, or perhaps developers assume that the HTML-development software that they are using auto-

Figure 7. Tabular output with option for CSV download.

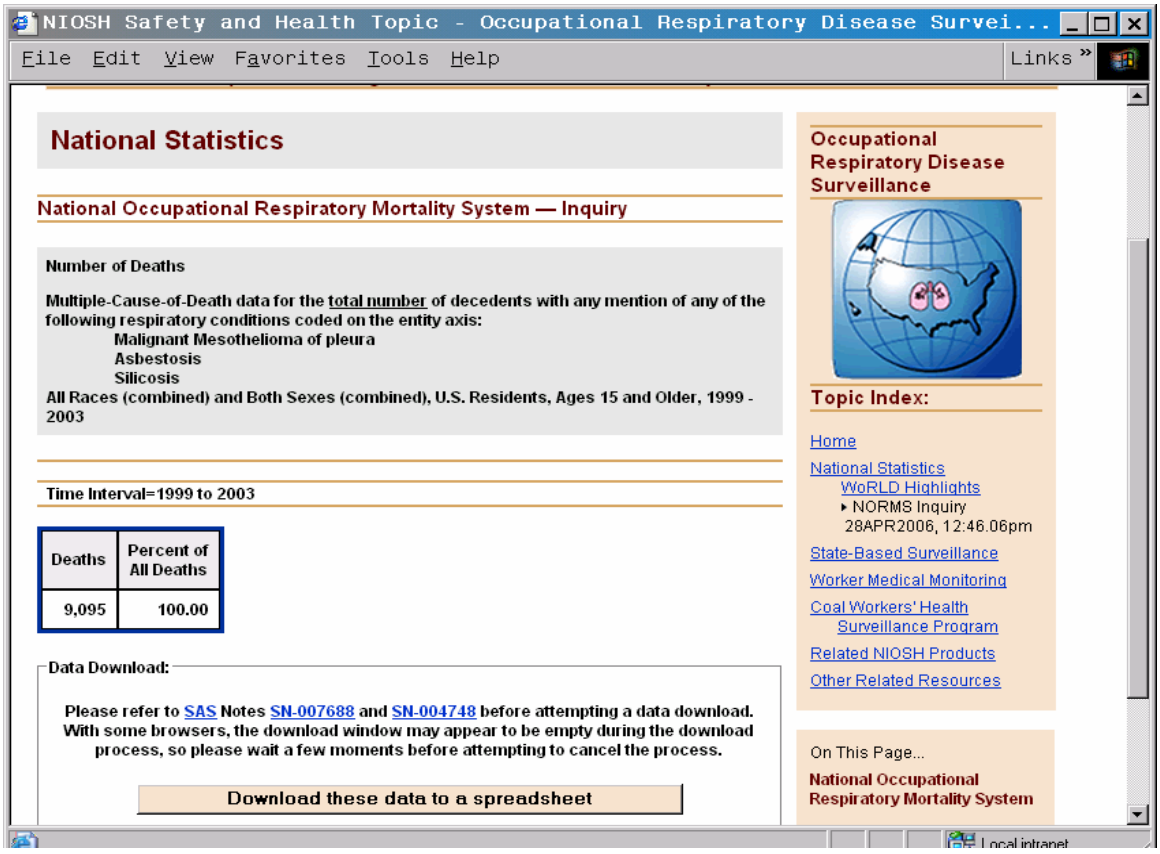


Figure 8. The data for populating the CSV file were temporarily saved at the same time that the output table was generated. The last DATA step creates the submit button shown at the bottom of Figure 7.

```
%MACRO CSV_FILE;

/** Create a temporary holding area for the current Dispatcher session
    in a reserved library called SAVE **/
%let rc=%sysfunc(appsrv_session(create));

data SAVE.freqs(keep = [ …list the variables that will populate the CSV file… ]);
     set freqs;
  run;
data _null_;
  file _webout;
  put '<FORM action="' "&_url" '">';
  put "<INPUT TYPE=HIDDEN NAME=_service  VALUE=&_service>";
  put "<INPUT TYPE=HIDDEN NAME=_server  VALUE=&_server>";
  put "<INPUT TYPE=HIDDEN NAME=_port  VALUE=&_port>";
  put "<INPUT TYPE=HIDDEN NAME=_program VALUE=path4.catalog02.csv.scl>";
  put "<INPUT TYPE=HIDDEN NAME=_sessionid VALUE=&_sessionid>";
  put "<INPUT TYPE=HIDDEN NAME=csvfile VALUE=norms_&hour_min_sec>";
  put 'Data Download:';
  put '[ …additional text and formatting shown in Figure 7… ]';
  put '<input type=submit value="Download these data to a spreadsheet">';
  put '</form>';
run;

%MEND CSV_FILE;
     %CSV_FILE;
```

Figure 9. The DS2CSV macro will be executed when you click on the download button shown at the bottom of Figure 7.

```
/*****************************************************/
/******  Contents of path4.catalog02.csv.scl  ********/
/*****************************************************/

MAIN:

submit continue;

/** Initialize the global macro variable called csvfile **/
   %global csvfile;

/** Use the DS2CSV macro to send a CSV file from the temporary holding area
             to the client's browser using a unique name [norms_&hour_min_sec].  **/
   %ds2csv(data=SAVE.freqs, conttype=y, contdisp=y, savefile=&csvfile..csv,
           csvfref=_webout, runmode=s, openmode=replace);
endsubmit;
RETURN;
```

Figure 10. Graphical output with mouseover and drill-down capabilities.

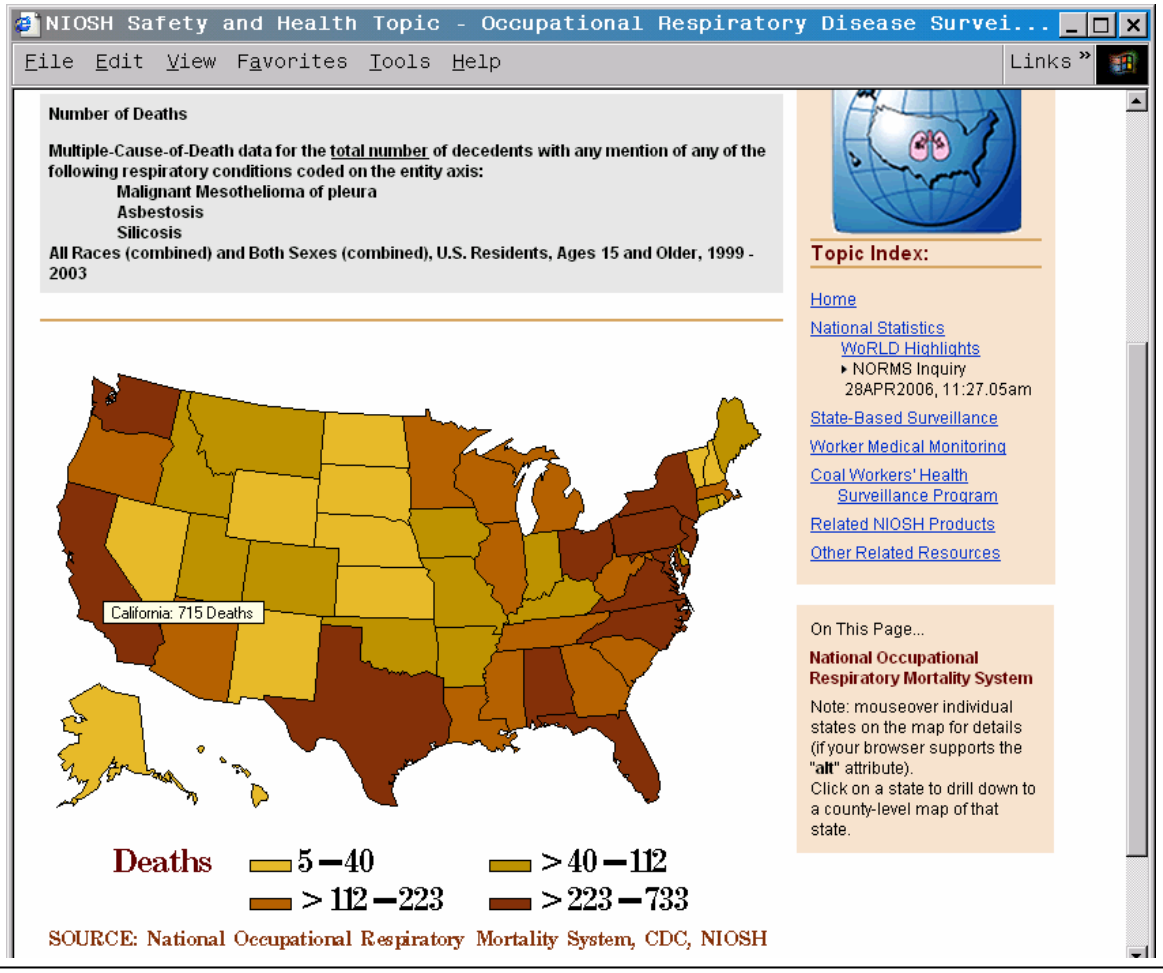


Figure 11. Code for creating the hypertext URL for each state on Figure 10.

```
submit continue;

proc template;
   define style template.norms;
    parent=styles.sasweb;
      replace TitleAndNoteContainer from Container /
         rules = NONE  frame = VOID  outputwidth = 100%  cellpadding = 1
         cellspacing = 1  prehtml = '<h3>'  posthtml = '</h3>';

   /* [ …other replace statements… ]  */

        end;   run;

ods listing close;
ods html body=_webout (dynamic  no_top_matter  no_bottom_matter)
         path=&_tmpcat (url=&_replay)  style=template.norms  rs=none;

data freqs;   set freqs;
if deaths=0 then do;
  VARSHOWN='Zero Deaths';
  Drill ='alt= " ' ||trim(left(fipnamel(state)))||": "||trim(left(VARSHOWN))||  ' " ';
       end;
else do;
  VARSHOWN=trim(left(put(deaths,comma11.)));
  Drill ='alt= " ' ||trim(left(fipnamel(state)))||": "||trim(left(VARSHOWN))|| " Deaths "|| ' " ';
       end;

     %macro multi_name_map;    /***** Assign field values for &disease1-&disease11 *****/
              %do m=1 %to &nd;
                   '&disease' || trim(left(&&m)) || '=' || trim(left(disease&&m)) ||
                %end;
      %mend multi_name_map;

           CoDrill =     'HREF="/path0/broker.exe'                      ||
                         '?_program=path4.catalog02.norms.scl'          ||
                         '&_service='        ||  symget('_service')     ||
                         '&' || 'state='     ||  trim(left(state))      ||
                         '&' || 'disease='   ||  trim(left("&disease1")) ||
                         '&' || 'disease0='  ||  trim(left(&nd+2))      ||
                         %multi_name_map
                         '&' || 'race='      ||  trim(left(&race))      ||
                         '&' || 'sex='       ||  trim(left(&sex))       ||
                         '&' || 'year_in='   ||  trim(left(&fr))        ||
                         '&' || 'year_out='  ||  trim(left(&to))        ||
                    '"'      ||
                         ' target="_blank" ' ||  trim(left(Drill))
                     ;

               call symput('whichmap',    'uscounty(where=(state=&state2))');
               call symput('whichid',     'state county');
               call symput('whichdrill',  'CoDrill');
 run;

 Proc Gmap data=freqs  map=maps.&whichmap  All;
   id &whichid;
   choro grpLegend / html=&whichdrill;
   run;   quit;

      ods html close;
   ods listing;
endsubmit;
```

matically corrects all 508 Accessibility errors. Testing services [10] are readily available for determining if Web pages comply with 508 Accessibility rules.

CONCLUSIONS

NORMS is a useful tool for implementation of a long-term NIOSH strategic goal of dissemination of occupational surveillance data for research purposes and disease prevention. The application provides direct, interactive access to National Vital Statistics data that can be used by industries, researchers, and state and local health departments to guide policies that may impact the respiratory safety and health of U.S. workers. The techniques discussed here work well for dynamically generating results from a large and complex national mortality database quickly and precisely. First, the data warehouse should be designed so that the application can process the data efficiently. Use where clauses in the initial DATA steps or procedure statements. Use macro subroutines as generic substitutes for specific procedural clauses and DATA-step options. Use compiled code for added security and efficiency. Combining the use of SAS/IntrNet with SAS macro and SCL coding is highly recommended for deploying a successful Web application.

REFERENCES

[1] National Center for Health Statistics. "Mortality - Multiple Cause of Death (Scientific Data Documentation)." <http://wonder.cdc.gov/wonder/sci_data/mort/mcmort/mcmort.asp> (Apr. 25, 2006).

[2] World Health Organization. 1992. *International Statistical Classification of Diseases and Related Health Problems 10th Revision, Volume 1*. Geneva, Switzerland: World Health Organization

[3] National Institute for Occupational Safety and Health. "National Occupational Respiratory Mortality System – ICD Codes." <http://webappa.cdc.gov/ords/norms-icd.html> (Apr. 25, 2006).

[4] SAS Institute Inc. "Application Dispatcher." <http://support.sas.com/rnd/web/intrnet/dispatch.html> (Apr. 25, 2006).

[5] Chisholm, W., G. Vanderheiden, and I. Jacobs. "HTML Techniques for Web Content Accessibility Guidelines 1.0." <http://www.w3.org/TR/WCAG10-HTML-TECHS/> (Apr. 25, 2006).

[6] Repole, W. 2001. SAS Web Tools: Advanced dynamic solutions using SAS/IntrNet® Software course notes. Cary, NC: SAS Institute Inc.

[7] SAS Institute Inc. "Supporting 508 Accessibility Requirements." <http://support.sas.com/faq/ts_qa/ods508.html> (Apr. 25, 2006).

[8] SAS Institute Inc. "Reference Documentation: GACCESSIBLE Macro." <http://support.sas.com/rnd/datavisualization/access508/macro-reference.html> (Apr. 25, 2006).

[9] SAS Institute Inc. "ACCESSIBLE Graphics option." <http://support.sas.com/rnd/datavisualization/access508/> (Apr. 25, 2006).

[10] Watchfire Corporation©. 2004. "WebXACT™ is a free online service that lets you test single pages of web content for quality, accessibility, and privacy issues." <http://webxact.watchfire.com/> (Apr. 25, 2006).

ACKNOWLEDGMENTS

SAS is a Registered Trademark of the SAS Institute, Inc. of Cary, North Carolina. WebXACT is a registered trademark of Watchfire Corporation.

The author would like to thank Norma McKee and John Barnett for technical assistance, as well as Ricki Althouse, Bob Castellan, Mike Attfield and Jacek Mazurek for guidance and support.

Disclaimer: The findings and conclusions in this abstract have not been formally disseminated by NIOSH and should not be construed to represent any agency determination or policy. Mention of the name of any company or product does not constitute endorsement by NIOSH.

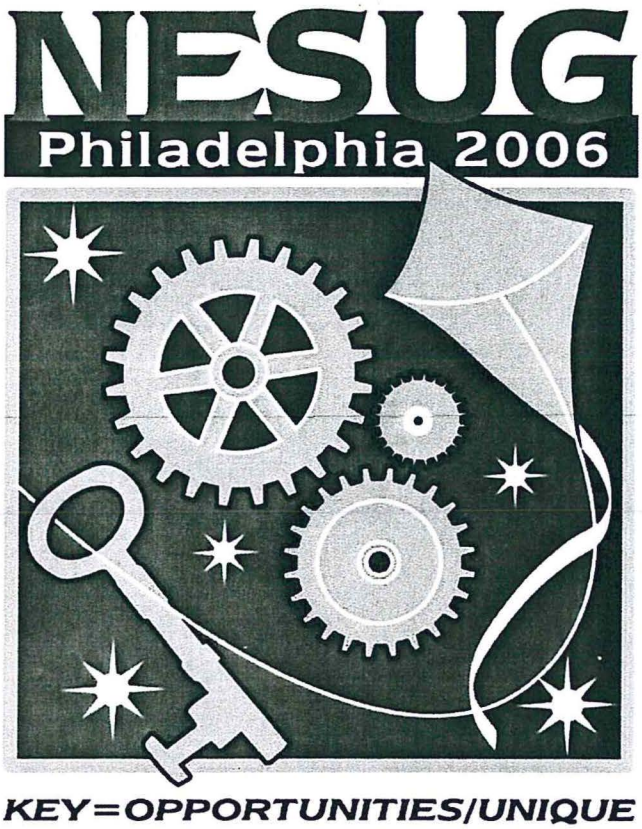
CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

John M. Wood
National Institute for Occupational Safety and Health
Division of Respiratory Disease Studies
1095 Willowdale Road
MailStop H_G900.2
Morgantown, WV 26505-2888
Work Phone: (304) 285-6159
Fax: (304) 285-6111
Email: JWood@cdc.gov

* *

NorthEast SAS Users Group Inc.
19th Annual Conference Proceedings



Philadelphia, PA
September 17-20, 2006

Conference Co-Chairs
Daphne Ewing and Paul Gorrell

Home
Contact
SAS:NESUG papers
SAS:PharmaSUG papers
SAS:PhUSE papers
SAS:SeUGI
SAS:SUGI papers
Search SAS-L
Browse SAS-L
SAS:RSS feeds

Fortune records
Marsh 1001
India
South America
UTC time
Virus info
My Links

Powered by SAS®
9.01.01M3P02162005.

The NorthEast SAS Users Group (NESUG) is a regional SAS® users group serving the northeast part of the country, from canada to Washington, DC, and west through New York and Pennsylvania.

The main purpose of the group is to promote the sharing of SAS® related information among members of the SAS® community.

NESUG meets this goal by conducting an annual conference, a Speaker Sharing program to local user groups, and by acting as an advocate for SAS® users to both SAS and SUGI.

This page provides an easy way to access the NESUG proceedings.

Google™ Search NESUG papers
25 Search results

You must use Adobe Acrobat Reader 5 or higher in order to view the *Proceedings*.
You can download the reader, for no charge, by clicking on the icon below.
Get Adobe Reader

Conferences

NESUG 2006	September 17-20, 2006, Philadelphia, PA
NESUG 2005	September 11-14, 2005, Portland, ME
NESUG 2004	November 11-14, 2004, Baltimore, MD
NESUG 2003	September 7-10, 2003, Washington DC
NESUG 2002	September 29 - October 2, 2002, Buffalo, NY
NESUG 2001	September 30 - October 3, 2001, Baltimore, MD
NESUG 2000	September 24-26, 2000, Philadelphia, PA
NESUG 1999	October 3-5, 1999, Washington, DC
NESUG 1998	October 4-6, 1998, Pittsburgh, PA
NESUG 1997	October 5-7, 1997, Baltimore, MD
...	
NESUG 1988	October 28, 1988, New York, NY