Paper 014-2008

Utilizing SAS® as an Integrated Component of the Clinical Research Information System

Christopher W. Schacherer, Clinical Data Management Systems, LLC
Jeffrey E. Gershenwald, University of Texas M. D. Anderson Cancer Center

ABSTRACT

The success of clinical research information systems in academic medical centers is often limited by the ability of analysts to access and extract data from the system or by the system's failure to support the operational aspects of the study. The current work describes an example of how SAS/Access®, user-defined formats, the SAS macro facility, and SAS e-mail distribution can help overcome these barriers and add value to existing clinical research information systems by facilitating data access and enhancing the efficiency of clinical research operations.

INTRODUCTION

The history of clinical research information systems within academic healthcare organizations is notable for research databases that vary widely in their sophistication. In one common scenario, individual investigators create a stand-alone disease registry (or, natural history databases) into which the variables of interest for their specific research program are collected by clinic staff, healthcare providers, and trainees. Early versions of these databases often had very little in the way of logical or referential integrity constraints, and commonly failed to have consistent data types within a given column. Moreover, these databases were frequently designed with little thought for how data would be analyzed or how the data system could facilitate the work of data managers, research assistants, and nurses in the conduct of research operations. Data were collected by one functional area, entered and managed by another, and analyzed by a third—each group being somewhat disconnected from the others due to their unique perspective on the data.

As robust relational database systems became more prevalent, data quality was bolstered by adding constraints on data elements and by referential integrity constraints between database tables. However, it often remained the case that some members of the research team saw interaction with the database system as little more than a necessary chore, others as an end in itself, and still others, by relying on data managers and programmers to extract the necessary data sets that would allow them to perform statistical analyses, did not interact directly with the system at all. Often, collaborators from different disciplines view the clinical research information system only from their unique perspective, and if systems are not designed to support the work of these different constituencies, the research program may fail to fully capitalize on the power of these centralized systems.

This paper presents one example of how SAS was used not only to enhance a clinical research information system by empowering the research team members, but also to engage the research team in maintaining timely, high-quality data.

THE CLINICAL DATABASE - MELCORE

The clinical research information system on which the current project was focused involved a natural history database used to collect and store information associated with 10,000+ patients diagnosed with melanoma who have agreed to participate in a prospective longitudinal study of their disease. The Melanoma Informatics, Tissue Resource, & Pathology Core database (MelCore) at the University of Texas M. D. Anderson Cancer Center is built on an Oracle 10g server and enables the web-based collection of study data through user-interface modules developed in Oracle Forms and Reports. The database contains tables for (among other things) patient demographics, lesion characteristics, treatments, treatment outcomes, and follow-up status. In addition, the database contains integrated modules that support a research tissue bank housing tissue and blood samples from the protocol participants.

SAS software was used: (1) to facilitate data extraction and formatting for biostatisticians and (2) as a work-flow management tool to improve clinical research operations. By supporting the work of study personnel involved in both statistical analysis and study operations, the integration of the solutions described below led to an increase in both team effectiveness and, importantly, data quality.

DIRECT ACCESS TO DATA

One of the main enhancements to the MelCore system was to utilize the native database connectivity available through SAS/Access to enable biostatisticians to connect directly to the database and retrieve data in real-time. In many similar clinical data management systems, either the security infrastructure is not mature enough to allow connectivity in a way that protects patient confidentiality by presenting data in a de-identified manner, or collaborators do not realize that direct database connectivity via their analysis tools is an option. In order to enable direct, real-time access to the study data, we began by creating a de-identified "biostatistician" view of the patient demographics table within the database system. This database view contained only system-generated patient identifiers that could be used to communicate with investigators and their clinical study team about individual patient records, but protected participant confidentiality. With this de-identified view of the data in place, direct access to the data was achieved by providing the technical solution via SAS software.

Prior to the implementation of this solution, analysts needed to request a data set from programmers or data managers and wait for the data set to be produced to their specifications. When working under tight timelines this created troublesome bottlenecks in the flow of data from the data collections operation through analysis and reporting and out to the investigator, sponsor, or oversight committee.

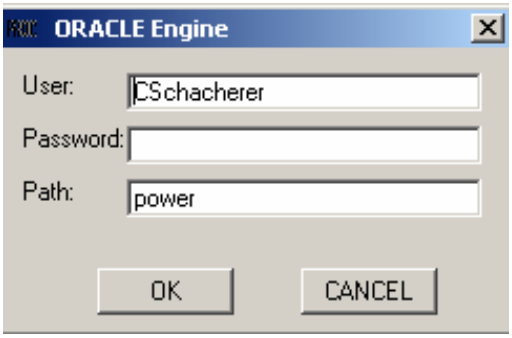
As described elsewhere (SAS Institute Inc., 1999, 2004), using native Oracle SQL*Net connections to the database, analysts can create SAS libraries that directly reference the database schemas to which they have access. The pre-requisites for such connections include having Oracle's SQL*Net client software installed on the server or workstation from which the user is running SAS as well as a copy of the "tnsnames.ora" file that contains the specific network references and connection parameters for the database(s) to which the user is trying to connect.

Having successfully configured Oracle's proprietary networking software, creating the library definition that connects SAS to the database can be accomplished as shown in the following example.

```
LIBNAME melpro ORACLE
   USER = CSchacherer
   DBPROMPT = YES
   PATH = "power"
   SCHEMA = melanoma_owner;
```

The preceding code creates a library named "melpro"; the "ORACLE" option specifies that the library is referencing an Oracle database. The "USER" option specifies that the connection will use the credentials of Oracle database user "CSchacherer", and the "PATH" specifies the entry within the tnsnames.ora file that points to the database instance to which the user wishes to connect. The "SCHEMA" option specifies the logical grouping of database objects to which this library is connecting.

One of the LIBNAME options in this example, "DBPROMPT", deserves special attention. When SAS attempts to attach to the database, the "DBPROMPT = YES" option forces the user to connect to the database using a graphical user interface (GUI) prompt for the Oracle username, password, and database (i.e., the "PATH"). For the melpro library example this prompt appears as follows:

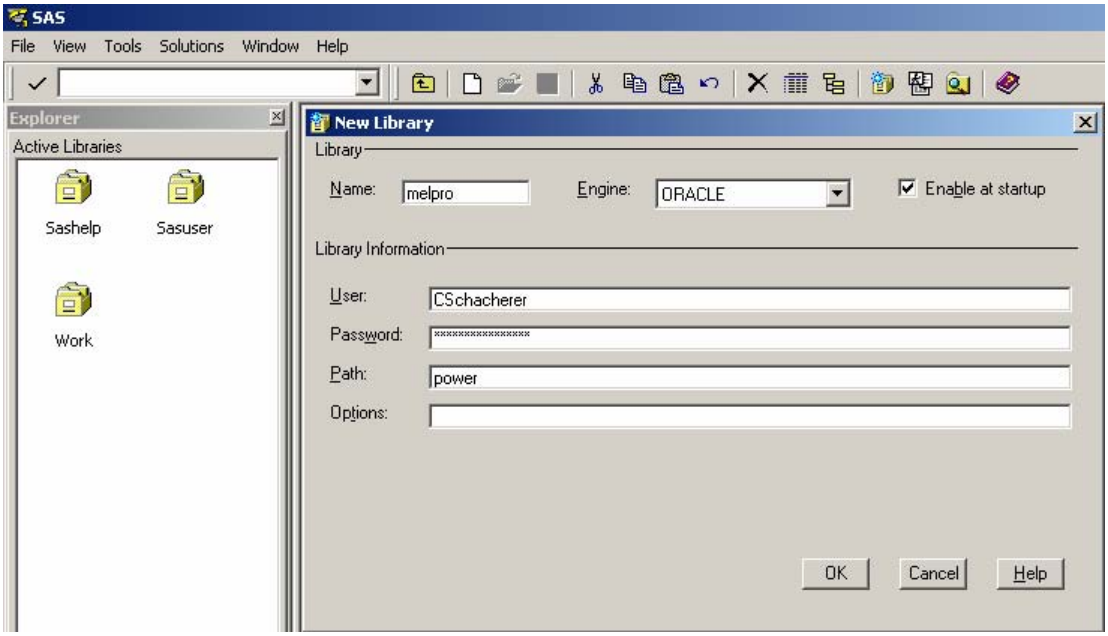


Alternatively, the database password can be hard-coded in your library specification using the password option, as in the following example.

```
LIBNAME melpro ORACLE
   USER = CSchacherer
   PASSWORD = mysecretpassword
   PATH = "power"
   SCHEMA = melanoma_owner;
```

However, hard-coding passwords in unencrypted files violates many organizations' security policies and is an especially dangerous practice in environments where code is shared among multiple users.

Finally, one could define the library in a manner that re-establishes the database connection each time the SAS system is started. To create such a library, right-click in the Explorer and choose "New". The "New Library" dialog box appears and allows you to choose the database engine and enter the necessary parameters for that particular database engine.



Regardless of how one establishes these connections, once these database-referencing libraries are established, the database tables and views to which the user has access can be used as source data for DATA step programming, SQL procedure queries, or analytic procedures (e.g., GLM, ANOVA, FREQ, etc.). The following example connects to the "smith_lab" database and creates the SAS data set "smith_biomarkers" from the data in the Oracle table "biomarkers" in the "smith_lab" SCHEMA.

```
LIBNAME smithlab ORACLE
   USER = CSchacherer
   DBPROMPT = YES
   PATH = "power"
   SCHEMA = smith_lab;

DATA work.smith_biomarkers;
 SET smithlab.biomarkers;
RUN;
```

Similarly, the following code creates a local SAS data set that extracts patient characteristics and biomarkers from both the "smith_lab" and "melpro" databases—joining the records on a common patient identifier:

```
PROC SQL;
 CREATE TABLE work.patient_biomarkers AS
   SELECT a.patient_id,a.gender,a.ethnicity,
          b.biomarker1, b.biomarker2
```

```
     FROM melpro.patients a, smithlab.biomarkers b
    WHERE a.patient_id = b.patient_id;
QUIT;
```

Once connected to the database(s), analysts have real-time access to all of the data to which they have been granted "select" (i.e., read) privileges in the database and can refresh their data sets as needed. Provided a comprehensive data dictionary for the project, analysts can write the code necessary to extract the individual data elements they need, create user-defined formats, and perform analyses.

Despite providing analysts with this much-needed autonomy, we found that because of the long learning curve inherent to the complex source data (and because this was the first time most of the analysts were using SAS to connect directly to a transactional database), analytic work was further facilitated by providing analysts with a SAS program that automatically extracts from the database only those data to which the analyst has "select" privileges and applies the appropriate user-defined formats to the extracted data. The first step in this process involves using the SAS macro facility (see Burlew, 2002 and Carpenter, 2004 for comprehensive technical guidance) to create a macro called %GETDATA.

%GETDATA

Within most enterprise database systems, there are internal system tables that control data access privileges. In the Oracle database system, the table that specifies the privileges granted to a specific user on a specific table is named "ALL_TAB_PRIVS". The %GETDATA macro uses this system table to: (1) determine the tables to which the "biostatistician" database role has access and (2) extract local copies of those tables as SAS data sets.

STEP 1. CREATE SOURCE TABLE

The "%GETDATA" macro begins by extracting from "ALL_TAB_PRIVS", the names of the tables to which the database role "biostatistician" has "select" privileges.

```
%MACRO GETDATA
PROC SQL;
 CREATE TABLE work.table_privs AS
   SELECT DISTINCT table_name AS oracle_name
     from melpro.all_tab_privs
    WHERE grantor = 'MELANOMA_OWNER' and
          grantee = 'BIOSTATISTICIAN' and
          privilege = 'SELECT';
QUIT;
```

The result of this step is the data set "table_privs"

VIEWTABLE: Work.Table_privs

	oracle_name
1	melview_patients
2	melcore_lesions
3	melcore_lesion_paths
4	melcore_followups
5	melview_event_survival

STEP 2. GENERATE SAS DATA SET NAMES

After creation of the data set "table_privs" (which contains the names of the Oracle tables and views to which "biostatistician" has "select" privileges), %GETDATA executes a DATA step that, for each record in "table_privs", creates a corresponding SAS data set name (sas_name) that will be used to create a local SAS data set containing the data in its associated Oracle table.

```
DATA work.table_privs;
 SET work.table_privs;
     i+1;
     ii=LEFT(i);
```

```
        sas_name=SUBSTR(oracle_name,9);
```

STEP 3. GENERATE AND VALUE MACRO VARIABLES

These data set names are then used to generate paired sets of macro variables (e.g., sas_name1 and oracle_name1, where ii = 1) that are assigned the values of the "sas_name" and "oracle_name" variables for record ii in the "table_privs" data set.

```
        CALL SYMPUT('sas_name'||ii,sas_name);
        CALL SYMPUT('oracle_name'||ii,oracle_name);
        CALL SYMPUT('total',ii);

   RUN;
```

The "total" macro variable is assigned a value corresponding to the number of records in the "table_privs" data set, and the following macro variables are created and assigned their values as follows:

sas_name(i)	sas_name(i) value	oracle_name(i)	oracle_name(i) value
sas_name1	patients	***oracle_name1***	melview_patients
sas_name2	lesions	***oracle_name2***	melcore_lesions
sas_name3	lesion_paths	***oracle_name3***	melcore_lesion_paths
sas_name4	followups	***oracle_name4***	melcore_followups
sas_name5	event_survival	***oracle_name5***	melview_event_survival

STEP 4. CREATE SAS DATA SETS

Finally, a DO loop is invoked to dynamically create the local data sets named in "table_privs" by executing a PROC SQL query of each of the tables to which the "biostatistician" role has "select" privileges.

```
%DO i=1 %TO &total;
PROC SQL;
 CREATE TABLE mel.&&sas_name&i AS
   SELECT *
     FROM melpro.&&oracle_name&i;
QUIT;

%END;
%MEND GETDATA
```

The first pass through this DO loop will resolve to the following query that creates a local copy of the melcore_patients data set called "patients" in the local library "mel".

```
PROC SQL;
 CREATE TABLE mel.patients AS
   SELECT *
     FROM melpro.melcore_patients;
QUIT;
```

The version of the data set "table_privs" produced after executing %GETDATA is presented below:

VIEWTABLE: Work.Table_privs

	oracle_name	i	ii	sas_name
1	melview_patients	1	1	patients
2	melcore_lesions	2	2	lesions
3	melcore_lesion_paths	3	3	lesion_paths
4	melcore_followups	4	4	followups
5	melview_event_survival	5	5	event_survival

Successful execution of %GETDATA in this example results in the five data sets—"patients", "lesions", "lesion_paths", "follow_ups", and "event_survival"—being produced in the local "mel" library. With these de-identified data sets, analysts can develop their statistical analysis and reporting programs and refresh the data later by running the %GETDATA macro.

Having been provided with local versions of the data sets, the analyst is still left with the task of creating user-defined formats to label values in the data sets.

This could be accomplished using the FORMAT procedure, as follows, to create the format "mel001_" in the mel library:

```
PROC FORMAT LIBRARY = mel;
  VALUE mel001_
        1 = 'Male'
        2 = 'Female'
        9 = 'Not Stated'
    OTHER = 'Incorrect Code';
```

However, given that the associated transactional system uses hundreds of value-label pairing groups (e.g., race, gender, yes-no-unknown, ulceration, clark-level, etc.) to characterize the data stored in the database, the task of manually coding user-defined formats would be very time-consuming. Moreover, requiring analysts to rewrite these value mappings in the form of user-defined formats introduces another possible source of error in reports and analytic output. Finally, as changes occur in the source database system, manually written formats risk becoming desynchronized from the clinical database. For example, new categories could be added for a given set of categorical variables and the change may not be communicated to the analysts. In order to avoid these problems, we implemented the %LOOKUPS macro inspired by Carpenter's (2004) %MKFMT macro to automate the creation of user-defined formats for the biostatisticians on the project.

%LOOKUPS

As was done in the %GETDATA macro, %LOOKUPS utilizes PROC SQL to extract data from the transactional system. Unlike %GETDATA, however, the data being extracted are not clinical data sets to be analyzed, but rather the data used by the data collection software to populate components of the user-interface. Within the MelCore system, the value-label pairs that allow a database user to select (for example) "Caucasian" from a drop-down list (and as a result store "0" in the "ethnicity" column of the "melcore_patients" table) are stored in a database table named "melcore_lookup_values". The records in this table are grouped into look-up "types" (e.g., 1, 2, 3, etc.) that link these records to their corresponding descriptive category in the table "melcore_lookup_types". Example data from these tables are shown below.

MELCORE_LOOKUP_TYPES		
Type	Shortname	Description
1	Yes/No	Dichotomous "yes/no".
2	Sex	Patient's sex.
521	Clark Level	Path. Rpt. Clark Level

MELCORE_LOOKUP_VALUES		
Type	Value	Label
1	0	No
1	1	Yes
2	1	Male
2	2	Female
2	9	Not Stated
521	1	Clark Level I
521	2	Clark Level II
521	3	Clark Level III
521	4	Clark Level IV
521	5	Clark Level V
521	910	Not Stated
521	980	Pending
521	999	Unknown

%LOOKUPS uses these data to produce user-defined formats in a manner that synchronizes the formats with the current lookup values in the transactional system. As new records are added to the lookup tables by the MelCore system administrator, re-execution of the %LOOKUPS macro creates the user-defined formats to reflect those

changes in the user-defined formats. The names of the user-defined formats produced by the macro correspond to the "look-up type" described in the Oracle table "melcore_lookup_types" (e.g., "mel001_" for look-up type "1 – Yes/No"). Therefore the analyst can simply run the %LOOKUPS macro and apply the user-defined formats to the data as described in the data dictionary provided to each analyst. For example, in order to apply the value-code pairs for look-up type "1" to the variable "tissue_consent_obtained", an analyst would apply the format as follows:

```
DATA test;
 SET work.patients;
     FORMAT tissue_consent_obtained mel001_.;
RUN;
```

As was done in the %GETDATA macro, the %LOOKUPS macro utilizes PROC SQL to extract data from the database and uses those data to dynamically create SAS objects. In %GETDATA, those objects were SAS data sets to be analyzed; in %LOOKUPS, the objects created are user-defined formats.

STEP 1. CREATE SOURCE TABLE

%LOOKUPS begins with extraction of a list of the unique look-up types stored in the application table "melcore_lookup_types". This table stores a description of all value-label pairing categories used to populate the user interface in MelCore. The resulting data set is presented below.

```
%MACRO LOOKUPS / STORE;

PROC SQL;
 CREATE TABLE formats.lookups AS
  SELECT DISTINCT type
    FROM melpro.melcore_lookup_types;
QUIT;
PROC SORT DATA = formats.lookups;
  BY type;
RUN;
```

VIEWTABLE: Formats.Lookups

	TYPE	SHORTNAME	DESCRIPTION
1	1	Yes/No	Dichotomous "yes/no" response.
2	2	Sex	Patient's sex.
3	521	Clark Level	Path. Rpt. Clark Level

STEP 2. GENERATE SAS DATA SET NAMES

Using the LOOKUPS data set as an input, the following DATA step creates a matched set of macro variables (lup<ii> and mel<ii>) corresponding to each record encountered in the "lookups" data set.

```
DATA formats.lookups;
 SET formats.lookups;
     i+1;
     ii=LEFT(i);
     mel_value = 'mel'||LEFT(type);
```

STEP 3. GENERATE AND VALUE MACRO VARIABLES

Each of these lookup types are then used to generate paired sets of macro variables (e.g., lup1 and mel1, where ii = 1) that are assigned the values of the "type" and "mel_value" variables for record ii in the "lookups" data set.

```
     CALL SYMPUT('lup'||ii,type);
     CALL SYMPUT('mel'||ii,mel_value);
     CALL SYMPUT('total',ii);
 END;
RUN;
```

In the current example, this step results in the following macro variables and their associated values:

lup(i)	lup(i) value	mel(i)	mel(i) value
lup1	1	***mel1***	mel1
lup2	2	***mel2***	mel2
lup3	521	***mel3***	mel521

STEP 4. CREATE LOCAL SAS DATA SETS

These macro variables are then utilized in the following DO loop to create data sets containing the value-label pairs that correspond to each look-up type.

```
%DO i=1 %TO &total;
PROC SQL;
 CREATE TABLE formats.&&mel&i AS
 SELECT value, label
   FROM melpro.melcore_lookup_items
  WHERE type = &&lup&i;
QUIT;
```

The first pass through this DO Loop resolves to:

```
PROC SQL;
 CREATE TABLE formats.mel1 AS
   SELECT value, label
     FROM melpro.melcore_lookup_items
    WHERE type = 1;
QUIT;
```

STEP 5. CREATE USER-DEFINED FORMATS

These data sets, in turn, are utilized within the same DO Loop to create the user-defined formats: mel001_, mel002_, and mel521_.

```
DATA formats.&&mel&i (RENAME = (value = START label = LABEL));
 SET formats.&&mel&i (KEEP = value label);
     FMTNAME='MEL'||PUT(&&lup&i,Z3.)||'_';
RUN;

PROC FORMAT CNTLIN = formats.&&mel&i LIBRARY = mel;
RUN;

%END;
%MEND LOOKUPS;
```

Once these formats are created, analysts can assign the appropriate format to each of the study variables as indicated in the data dictionary and be assured that the formats used in their programs are synchronized with the transactional system. Every time the %LOOKUPS macro is executed, the user-defined formats are resynchronized with the data collection system. This removes a source of potential error in analytic output and saves analysts considerable time in writing code to create user-defined formats.

From the standpoint of reduction of resources necessary to conduct clinical research, however, the SAS solution that had the greatest overall impact on the research program was an e-mail notification system utilized to make more efficient use of research nurse resources.

AUTOMATED E-MAIL NOTIFICATION SYSTEM

Prior to implementation of this system, research nurses logged into multiple computer systems on a daily basis and manually cross-referenced clinic schedules against lists of protocol participants in order to identify protocol

participants from whom they had not yet collected protocol-specified blood samples. In order to reduce this burden, an e-mail notification system was developed for automating this function. The SAS program used to automate this task was scheduled to run every morning and e-mail staff with the lists that they previously produced using a manual process. The following code, although limited to this specific task, could easily be expanded to support other similar tasks such as identifying patients from whom the research nurse needs to obtain an updated informed consent or to identify patients that met protocol-specific eligibility requirements. Of course, in order to implement these solutions, the necessary data needs to be captured in the clinical research information system supporting the study. For an in-depth treatment of the topic of e-mail delivery of SAS output, see Page (2004).

STEP 1. VALUE MACRO VARIABLE AND PREPARE REPORT

A macro variable (blood_draws) is initially assigned the value of "0"; this macro variable will be used later in the program to determine which of two e-mails will be sent—one indicating that there are no protocol patients for whom blood-draws are needed on the current day or one providing a list of the appointment details for protocol participants from whom a blood sample should be collected.

```
%LET blood_draws = 0;
```

After the assignment of "0" to the macro variable "blood_draws", the Output Delivery System (see Haworth, 2001; SAS Institute Inc., 2004) is used to generate an HTML document that will be used as a worksheet for the research nurse in coordinating blood draws for the current day. The ODS command initiates the creation of the HTML file "proj5_draws.html" and the output produced in this file will be titled with the current day's date.

```
ODS HTML BODY ='c:\proj5_draws.html';

TITLE 'Project 5 Blood Draws for '"%SYSFUNC(DATE(),WORDDATE.)";
```

Then the following PROC SQL query is run to assign "blood_draws" a value equal to the number of scheduled patient visits for the current day that involve participants on a specific protocol for whom the study team has not yet completed drawing the blood samples required by the protocol. This query links to both the study database and to the appointment scheduling system and dynamically builds the WHERE clause in order to query the hospital scheduling system for the current day's schedule.

```
PROC SQL NOPRINT;
   SELECT COUNT(a.mrn) INTO :blood_draws
      FROM melpro.melcore_patients a,
           melpro.melcore_protocol_participants b,
           melpro.mellink_patsched_daily c
     WHERE c.appt_date = PUT(YEAR(DATE()),Z4.)||
           PUT(MONTH(DATE()),Z2.)||PUT(DAY(DATE()),Z2.) and
           a.patient_id = b.patient_id and
           a.mrn = c.mrn and
           b.blood_draws_complete = 0 /*blood draws not completed*/ and
           b.protocol = 5;
QUIT;
```

Next, another PROC SQL query (using the same WHERE clause as the previous query) is performed to generate a list of the appointment details for protocol participants with clinic visits scheduled for the current day. This list will form the body of the HTML report being generated by ODS. For cases in which "blood_draws" is assigned the value of "0", the body of the HTML file "proj5_draws.html" will be blank except for the TITLE.

```
PROC SQL;
   SELECT a.mrn,c.appt_rpt_to_loc AS location,
          c.actv_desc as activity, c.appt_time AS TIME
     FROM melpro.melcore_patients a,
          melpro.melcore_protocol_participants b,
          melpro.mellink_patsched_daily c
    WHERE c.appt_date = PUT(YEAR(DATE()),Z4.)||
                        PUT(MONTH(DATE()),Z2.)||PUT(DAY(DATE()),Z2.) and
          a.patient_id = b.patient_id and
          a.mrn = c.mrn and
```

```
          b.blood_draws_complete = 0 /*blood draws not completed*/ and
          b.protocol = 5
 ORDER BY c.appt_time;
QUIT;
ODS HTML CLOSE;
```

STEP 2. CREATE EMAIL FILENAME

At this point, the ODS HTML output file is closed, and the e-mail that delivers the needed information to the research nurses is prepared. The e-mail framework is prepared by defining a FILENAME (“bloods”) of type EMAIL. The specifications of this FILENAME type include the e-mail interface being utilized to send the message as well as the e-mail address from which it will be sent.

```
FILENAME bloods EMAIL
   EMAILSYS  = mapi
   EMAILID   = 'meladmin@mdanderson.org';
RUN;
```

STEP 3. GENERATE AND SEND E-MAIL MESSAGE

Once the message framework is specified, a DATA _NULL_ step is performed that will conditionally write and send one of two different e-mails based on the value of the macro variable “blood_draws”. If there are no patients participating on the specified protocol who are scheduled for a clinic visit on the current day (i.e., blood_draws = 0) a message will be sent to the research nurse and data manager with the subject line “No Priority Blood-Draws Today”. The body of this message will remain blank; the research nurse simply sees this subject line and knows that there is nothing more to do on this project for today with respect to specimen collection. If protocol participants are identified from whom the research nurses will need to collect bloods, a different message is prepared—one that includes the research nursing supervisor and to which is attached the list of appointments involving protocol participants—“proj5_draws.html”. The latter e-mail also contains a message in the body of the e-mail directing the research nurse to perform additional checks on the blood draw records as they pertain to this particular protocol.

```
DATA _NULL_;

 FILE bloods;
      IF &blood_draws = '0' THEN
        DO;
          PUT '!EM_SUBJECT! No Priority Blood-Draws Today';
          PUT '!EM_TO! (MelNurse1@mdanderson.org datamgmt1@mdanderson.org)';
          PUT '!EM_CC! (MelNurse2@mdanderson.org)';
        END;
      ELSE
        DO;
         PUT '!EM_SUBJECT! Blood-Draws for Project 5 (see attachment) ';
         PUT '!EM_TO! (MelNurse1@mdanderson.org)';
         PUT '!EM_CC! (MelNurse2@mdanderson.org ResNurseSupv@mdanderson.org
                       datamgmt1@mdanderson.org datamgmt2@mdanderson.org)';
         PUT '!EM_ATTACH! c:\proj5_draws.html';
         /*Message in body of e-mail*/
         PUT 'The attached patient list contains patients for whom it is a
              priority to draw blood samples under protocol XXX-123 for Project 5.
              These patients may need to be consented/reconsented on the
              protocol.';
         PUT ' ';
         PUT 'Please also check your records for these patients to make sure that
              the 40ml maximum blood draw for each patient has not been met.';
END;

RUN;
```

Having this e-mail scheduled for delivery every morning, the research nurses are saved significant time that was previously spent manually cross-referencing multiple systems—clinic scheduling, protocol enrollment, and their own blood-draw records.

CONCLUSION

The illustrations in this paper demonstrate how the integration of SAS functionality into a clinical research information system can facilitate the work being performed by study personnel from different disciplines. Utilization of native database connectivity, combined with the dynamic extraction and utilization of data from the data collection system allowed our study statisticians to focus on analytic work instead of developing and maintaining code used to perform data extraction and formatting tasks. Similarly, by automating the otherwise manual task of cross-referencing multiple data sources to generate protocol-specific task lists, research nurses were allowed to focus on the conduct of the study instead of needlessly redundant tasks. One unanticipated benefit of these solutions was the enhancement of the study team's appreciation of (and interest in) the data collection system and, more importantly, the quality of the data stored in it. As the data collection system evolved to provide support for individuals working on the study, study team members took a very active interest in data collection activities and adherence to data collection standard operating procedures. For example, the quality of blood-draw data entered in the system was now of significant interest to all of the research nurses involved in the study. In order for the time-saving benefits of the system to be realized, blood-draw data had to be entered in a consistent and timely manner by all members of the research team. Finally, the common reliance on the clinical data management system fostered by the previously described enhancements produced the ancillary benefit of helping team members understand one another's contributions to the project and provided a commonly understood language for discussing the various data elements.

REFERENCES

Burlew, Michelle M. (2002). SAS Macro Programming Made Easy. Cary, NC: SAS Institute Inc.

Carpenter, Art (2004). Carpenter's Complete Guide to the SAS Macro Language, Second Edition. Cary, NC: SAS Institute Inc.

Haworth, Lauren E. (2001). Output Delivery System: The Basics. Cary, NC: SAS Institute Inc.

Page, Jacues (2004). Automated distribution of SAS results. Proceedings of the SAS User's Group International 29th Annual Meeting. Quebec, QC.

SAS Institute Inc. (2004). SAS 9.1 SQL Procedure User's Guide. Cary, NC: SAS Institute Inc.

SAS Institute Inc. (2004). SAS 9.1 Output Delivery System: User's Guide.

SAS Institute Inc. (1999). SAS/ACCESS Software for Relational Databases: Reference, Version 8. Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1999). Procedures Guide, Version 8. Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1989). SAS Guide to the SQL Procedure: Usage and Reference, Version 6, First Edition. Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

This work was supported by National Cancer Institute grant P50 CA093459.

CONTACT INFORMATION

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

Christopher W. Schacherer, Ph.D.
Clinical Data Management Systems, LLC
6666 Odana Road #505
Madison, WI 53719
Phone: 608.630.2637
E-mail: CSchacherer@cdms-llc.com
Web: www.cdms-llc.com