SAS® Techniques to Get the Most out of Oracle Clinical®

David Johndrow, Analytical Sciences, Inc., Durham, NC

ABSTRACT

Many research companies are short on Oracle talent but long on SAS expertise. How can such organizations further benefit from SAS experience within the clinical setting? This presentation has 3 main goals:

1. Expose an alternative approach to the extraction of clinical data into SAS (with a larger role played by SAS programmers). This approach can also potentially have a significant impact on cost.
2. Explore how SAS can be employed to tap into the underpinning tables of the Oracle Clinical system, and how this available treasure trove of data can be used within the SAS setting to generate project reports and track study progress.
3. Expand on the ability to create SAS reports by demonstrating some of the newer output formats available in SAS, via the Output Delivery System (i.e. RTF, HTML). A few simple SAS tricks can turn reports into interactive tools.

ALTERNATIVE APPROACH TO ORACLE CLINICAL DATA EXTRACTION

TYPICAL APPROACH

The typical approach to extract data into Oracle Clinical (OC) uses OC to ***push*** the data to SAS. That is, the process is entirely initiated within OC. SAS is invoked on a server, and the study dataset views are called via SAS.

ALTERNATIVE APPROACH

An alternative approach, however, uses SAS to ***pull*** the data from Oracle Clinical. This process is initiated entirely within SAS and has several key advantages.

TOOLS NEEDED

In order to use the alternative method, you must have the following three tools:

1. SAS installed on a workstation.
2. A SQLNET connection to Oracle from the same workstation.
3. SAS Access to Oracle Databases.

Typically, if you have access to Oracle Clinical, and you wish to create your SAS datasets from OC, you already have access to the first two items, thus the third item is probably all you lack.

EXTRACT METHOD (INFORMATION)

Before we write any code to demonstrate how to extract OC data directly from SAS, we must first gather some information that speaks to how we connect to Oracle Clinical, where we pull the data from, and where we will put the resultant SAS datasets.

We will connect to the Oracle database via SAS using the same account information you use to connect to Oracle Clinical:

Item:	Example:
Your OC Username	OPS$DAJ
Your OC Password	xxxxx
Your OC Instance	PRD01

We will need to decide where we will be placing the datasets we are extracting. In this example, let's assume that a SAS library called ASI01Stable has already been created, and we wish to create a Demographics dataset:

Item:	Example:
Resulting SAS data library	Asi01Stable
Resulting SAS dataset name	Demographics

Lastly, we need to gather information about the study, view types and views that we will be extracting. Assuming your study is called ASI01; let's use the following examples:

Item:	Example:
OC Study Name	ASI01
OC View Type	Stable
OC View Name	Demog

EXTRACT METHOD (PROGRAM EXAMPLE)

The following code example uses SAS Proc SQL to connect to the Oracle database, query against the view you specified, and save this data to the SAS dataset of your choosing:

```
proc sql;
   connect to ORACLE (user="OPS$DAJ"
                      password=xxxxx
                      path="@PRD01");
   create table ASI01STable.Demographics as
     (select * from connection to ORACLE
       (select * from ASI01$STABLE.DEMOG));
quit;
```

EXTRACT METHOD (FURTHER SUGGESTIONS)

1. Write a SAS macro to handle the importation of all Oracle Clinical data, then you need only change parameters. Example:

```
%getOCdata(OPS$DAJ,xxxxx,PRD01,
           ASI01Stable,Demographics,
           ASI01,Stable,Demog);
```

(See appendix program 1 for code examples)

2. Develop a tool to collect the OC connection information separately from the SAS program. This will avoid inadvertent saving of sensitive data within SAS programs. This tool can be called by the SAS program.

EXTRACT METHOD (ADVANTAGES)

Cost: Does not require that you have SAS on the OC server. SAS Access to Oracle cheap in comparison to server version of SAS.

Speed: Running SQL directly against Oracle eliminates the OC PSUB overhead and generally runs much faster.

Recyclable: With a macro, only the details need changing for each new dataset, study, user, etc.

Freedom: SAS has long been free of the 8-character variable naming convention. OC has been slow to catch up.

USE SAS TO ACCESS OC SYSTEM TABLES

THE SETTING

There is good news in that OC already provides many standard reports that you can access and run, in regards to your studies and the database in general.

However, the bad news is that these reports are *very* general in nature, and not very useful for concisely tracking study progress.

What does this all mean? Companies are pretty much left to their own devices to generate customized study progress reports, or present meaningful metrics in a way that is of any use.

Given that you are on your own, it comes as more bad news that in order to create reports, you will need to gain access to, and familiarity with, some of the OC system tables. These tables can be rather difficult to work with, since the Oracle Clinical system has at its core, a very highly normalized database architecture.

Help is on the way, however, because it comes as good news that you can focus on a select few OC system tables and generate very useful project tracking reports in SAS.

OC SYSTEM TABLES

Let's focus on 8 OC system tables for the remainder of this paper. With these 8 tables, a myriad of extremely useful reports can be created with just a basic understanding of what is contained within these tables, combined with a little SAS know-how. The 8 tables of interest are as follows:

Clinical_Studies: Study names and identifiers
OCL_Sites: Site name and identifiers
OCL_Investigators: Investigator names and identifiers
Patient_Positions: Patient study & OC internal identifiers

Data_Clarification_Forms: DCF details
Discrepancy_Entries: Discrepancy database details
DCMs: Generic DCM information
Received_DCMs: Detailed logged in and received DCM info.

Use SAS to create dataset versions of each of the OC system tables (using an extract method just like that used in the example above). Update these SAS datasets after every batch validation.

Disclaimer: In order to do this, you MUST work with your DBA to create an OC access account similar to the Oracle RXC account. This action should comply with your organization SOPs and the account should be set up as read-only! For this demonstration, let's assume an account called RXCREAD was created for this purpose.

OC SYSTEM TABLES (PROGRAM EXAMPLE)

First, you should create SAS copies of the needed OC system tables. From that point forward, you will be working against the SAS dataset copies of these system tables, which is far safer than working against the tables themselves and will amount to less burden on the Oracle Clinical system. We have already introduced the concept in a previous example, however, with later versions of SAS, you can achieve the same goal in a much simpler way using the libname statement. Here is a streamlined alternative to the proc sql approach:

```
Libname OCClin 'c:\whatever\OC tables';
Libname RxcR oracle user="rxcread"
                    password=xxxx
                    path="PRD01";

data OCClin.Clinical_Studies;
  set RxcR.Clinical_Studies;
run;
```

CUSTOM REPORTS (THE PROCESS)

In order to build custom reports, the examples in this paper will demonstrate how you can write SAS code to reconcile your study data (as it comes out in the actual data extracts) with the OC system data (which reports on the status of data and data discrepancies within the system). These two bodies of data, when reconciled, can be used to draw a comprehensive picture of study progress.

CUSTOM REPORTS (EXAMPLE ONE)

For our first and simplest example, let's suppose that we want to track the number of data clarification forms that are "out-the-door or back" for a given study. That is, either sent, received or closed.

We will need to access the **Clinical_Studies** and **Data_Clarification_Forms** tables.

In order to create a report of this type, here are the logical steps we must take:

1. Retrieve the DCFs from table Data_Clarification_Forms, using clinical study identifiers found in Clinical_Studies.
2. Perform a SAS frequency on these types, transpose the results.
3. Adjust some values to be cumulative (since closed DCFs were at one time sent & received!)

Now that we have the pseudo-code necessary to report the relevant information, how might we present it? Consider the following table:

DCF Status	Last Report	This Report	Cumulative	Outstanding
Sent	12	20	32	**8**
Received	11	13	24	
Closed	10	8	18	

In the example above, we used the program output from the current status (retrieved from OC system tables only, not study data), and we used the current data bundled with older data (last week, month, etc.) to tell the story of activity for Data Clarification Forms.

This example shows that we currently have 32 reports that have been sent out, of these, 24 have been returned in-house and of those, 18 are closed. The 2nd column demonstrates the same numbers as of the last reporting period, where the 3rd column exhibits activity that has obviously occurred since (subtract column 2 from column 4). Lastly, the "Outstanding" column is included because when it get right down to it, this is what investigators or project managers want to know!

(See appendix program 2 for code examples)

CUSTOM REPORTS (EXAMPLE TWO)

For our next example, let's suppose that we want to create a patient grid, detailing the status of all CRF pages. This would be a useful tool to monitor study progress.

For this report, we will need to reconcile study data (obtained from data extracts) with what OC knows about each CRF page. Let's use a very small, 4-page CRF as an example. Suppose we decided to depict study progress using the following grid, using that 4-page CRF, with 2 visits, and 2 sites:

Protocol XYZ		CRF Pages			
		Screening		Week 1	
PT	SITE	Cover	Demog	Lab	QOL
001	1101-Lagle				
004	1345-Lin				
Etc.					

All you need to do to create this complete story, is to go through the following programmatic steps:

Step 1: Generate a similar grid, based on your extract data. That is, what data you have coming out in an extract, use it to denote what CRF pages are found. Populate each found page with a 'C' for complete. Let's assume our findings were:

PT	SITE	Cover	Demog	Lab	QOL
001	1101-Lagle	C	C	C	C
002	1101-Lagle	C	C		
003	1101-Lagle	C	C	C	C
004	1345-Lin	C		C	
005	2020-Hilton	C	C	C	

As you can see, patients 001 and 003 have all expected data.

Step 2: Similarly to what we did in step 1, we want to build a grid denoting found CRF pages. This time, however, we are going to build our construct based on what OC knows in regards to the existence of these pages. In order to do this, we will need to pull data from the OC system table called **Received_DCMs**. You will need to do some merging with other tables, such as **Clinical_Studies**, to know that you are pulling information relevant to only this study.

In addition to constructing a grid, let's pay attention to the information in the Received_DCMs table regarding the status of this particular DCM. Let's group status under 'R' for 2nd-pass reconciled, 'D' for undergoing data entry and 'L' meaning the page is logged into the system. Having done so, we might come up with a grid that looks like this:

PT	SITE	Cover	Demog	Lab	QOL
001	1101-Lagle	R	R	R	R
002	1101-Lagle	R	R	D	D
003	1101-Lagle	R	R	R	R
004	1345-Lin	R		R	L
005	2020-Hilton	R	R	R	

Step 3: Simply overlay the two grids! Let the 'C' values in the study data grid override the 'R' values in the system table grid. After all, reconciled data should now be coming out in your stable extracts, otherwise you have a problem! After the merge, your table, with a few header columns, can now look like this:

Protocol XYZ		CRF Pages			
		Screening		Week 1	
PT	SITE	Cover	Demog	Lab	QOL
001	1101-Lagle	C	C	C	C
002	1101-Lagle	C	C	D	D
003	1101-Lagle	C	C	C	C
004	1345-Lin	C		C	L
005	2020-Hilton	C	C	C	
C=Complete * D=Data Entry * L=Logged into Oracle Clinical					

This new table can now be a very useful tracking tool in the hands of a data manager. You are essentially tracking your study data as it moves through your OC and SAS systems.

CUSTOM REPORTS (EXAMPLE THREE)

For our third custom report example, let's assume we want to see our study progress data in a more aggregate manner. For example, we would like to see CRF pages grouped into visits, and rather than detailed patient listings, counts provided by site.

Let's also assume that we'd like to speak to the cleanliness of the data, something we were ignoring in the last program example.

The good news once again is, we have already done most of the hard work in example 2! We need only reconcile that data with OC system data related to discrepancies, then summarize and rearrange the data. Based on the same data and CRF from the previous example, let's shoot for the following report:

Site	Patients Logged into OC	Screening	Week 1	Complete & Clean Patients
1101-Lagle				
1345-Lin				
2020-Hilton				
All Sites	0	0	0	0

From this table, you can see that we plan to populate counts for each site, based on how many patients that site has at each status. Let's follow the logic necessary to complete such a report:

Step 1: Use the clinical table **Discrepancy_Entries** to determine which patients have discrepancies.

Step 2: Write SAS code to calculate the aggregate counts of complete visits, by site (however you define a visit). Do the same for a total across all visits (even though you will not be showing this in the final table, you will use it in step 3). This counting can occur across the resultant table in program example two above.

Step 3: Subtract the number of "unclean" patients from the totals column you created in step 2. This will determine the number to place in the final column.

Referring back to the table we yielded in custom report example two, notice the areas shaded below as areas where the visit is complete for that patient:

Protocol XYZ		CRF Pages			
		Screening		Week 1	
PT	SITE	Cover	Demog	Lab	QOL
001	1101-Lagle	C	C	C	C
002	1101-Lagle	C	C	D	D
003	1101-Lagle	C	C	C	C
004	1345-Lin	C		C	L
005	2020-Hilton	C	C	C	
C=Complete * D=Data Entry * L=Logged into Oracle Clinical					

We can see that our new table would be populated as follows. Note that the number of complete and clean for site 1101 is only 1 (shaded). Evidently, our program detected that 1 of the 2 complete patients for this site had an outstanding OC discrepancy!

Site	Patients Logged into OC	Screening	Week 1	Complete & Clean Patients
1101-Lagle	3	3	2	1
1345-Lin	1	0	0	0
2020-Hilton	1	1	0	0
All Sites	0	0	0	0

USE SAS ODS TO ENHANCE CUSTOM REPORTING

For the examples used in this presentation, we generated output SAS datasets that we then exported to Microsoft Excel for a more polished look.

If you have the appropriate version of SAS, the SAS Output Delivery System (ODS) makes many other output formats viable:

1. Web (HTML)
2. Rich Text Format (RTF)
3. Postscript (PS)
4. Portable Document Format (PDF)
5. Future - Extensible Markup Language (XML)

Many papers have been published exploring the new power of the ODS, and it is not the intent to attempt to explore the various features of this system within this paper. However, it is a worthwhile endeavor to introduce a basic data manipulation trick that can enable you to build reporting documents that are actually hyperlinked, using the HTML output available from the ODS.

First, as you are no doubt already aware, it is painfully simple to redirect SAS output to an HTML web page. The following code example demonstrates how to create HTML code. For this example, we will call PROC PRINT to output the resultant SAS dataset from our custom report program example three above.

```
ods html body='c:\…\ODSOverall.html';

proc print data=overallstats noobs;
  title 'Protocol XYZ - Overall Study
Status';
  footnote;
run;

ods html close;
```

The output from this program would look similar to the Microsoft Excel spreadsheet examples presented before:

Site	Patients Logged into OC	Screening	Week 1	Complete & Clean Patients
1101-Lagle	3	3	2	1
1345-Lin	1	0	0	0
2020-Hilton	1	1	0	0
All Sites	0	0	0	0

Boring! You can get much more creative! Why not try designing your programs so that users can drill-down into their data?

For example, before we issue the proc print statement, use our SAS dataset to create a string variable with standard html anchors around itself.

By doing this, we create data that will in turn function as a hyperlink in our resulting output. Consider the following program example, paying special attention to the extra string information wrapped around the site variable below:

```
data OverallStats2;
  attrib Site length=$64.;
  set OverallStats;
  if site ^= 'All Sites' then Site =
    '<A HREF="' || compress(site) ||
    '.html">'  ||  compress(Site)  || '</A>';
run;
```

So now if we redirect Proc Print output to an HTML page via the ODS, we get pretty much the same table with one very important exception: Notice that the sites are now hyperlinked! One click of the mouse on each site could take the user to an entirely different web page, detailing more specific information relevant to that site alone!

Site	Patients Logged into OC	Screening	Week 1	Complete & Clean Patients
1101-Lagle	3	3	2	1
1345-Lin	1	0	0	0
2020-Hilton	1	1	0	0
All Sites	0	0	0	0

(See appendix program 3 for code examples)

CONCLUSION

We can use SAS to directly gain access to our study data in a way that is more economical and far less restrictive than the traditional approach.

Beyond study data, there is a wealth of data in the Oracle Clinical system tables that can be accessed and manipulated utilizing SAS.

SAS, via the ODS, now provides us with many more effective tools which can be used to present study data or reports.

CONTACT INFORMATION

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

David Johndrow
Analytical Sciences, Inc.
2605 Meridian Parkway, Suite 200
Durham, NC 27712
Work Phone: (919) 544-8500
Fax: (919) 544-7507
Email: djohndrow@asciences.com
Web: www.asciences.com

APPENDIX PROGRAM EXAMPLE 1

```
/************************************************
* Program Name: PharmaSUGExample1.sas
* Author:       David Johndrow,
*               Analytical Sciences, Inc.
* Purpose:      Demonstrate a macro that can be used to
*               extract Oracle Clinical study data via
*               a direct connection through SQLNet,
*               using SAS Access to Oracle.
************************************************/

%macro GetOCData(OCUser,    /* OC user name */
           OCPw,      /* OC password  */
           OCPath,    /* OC database  */
           SASLName,  /* SAS destination library   */
           SASDName,  /* SAS destination dataset   */
           StudyName, /* OC study name             */
           ViewType,  /* Type of view (Stable, etc.*/
           ViewName); /* View name                 */
  proc sql;
    connect to ORACLE (user="&OCUser"
                       password=&OCPw
                       path="@&OCPath");
    create table &SASLName..&SASDName as
      (select * from connection to ORACLE
        (select * from
          &StudyName.$&ViewType..&ViewName));
  quit;
%mend GetOCData;

/* Example call */
%GetOCData(OPS$DAJ,xxxxx,PRD01,
           ASI01STable,Demographics,
           ASI01,STABLE,DEMOG);
```

APPENDIX PROGRAM EXAMPLE 2

```
/*****************************************************
* Program Name:  PharmaSUGExample2.sas
* Author:        David Johndrow,
*                Analytical Sciences, Inc.
* Purpose:       Tap into 2 Oracle Clinical tables to
*                retrieve a report about the status of
*                data clarification forms.  The
*                library "clin" below assumes you have
*                already extracted the needed OC
*                system tables to SAS datasets.
*****************************************************/
%macro BuildDCFs(StudyName);

  /* Set a macro variable for the study identifier */
  data _null_;
    set clin.clinical_studies
      (keep=clinical_study_id study);
    if study="&StudyName";
    call symput("STUDY_ID",clinical_study_id);
  run;

  /* Retrieve all data clarification forms for
     the study, using the identifier just retrieved */
  data curdcfs;
    set clin.data_clarification_forms
      (keep=current_status clinical_study_id);
    if clinical_study_id=&STUDY_ID;
  run;

  /* Get freq-y with it. */
  proc freq;
    tables current_status / out=dcfs;
  run;
  data dcfs;
    set dcfs;
    if current_status='SENT' then status=1;
    if current_status='RECEIVED' then status=2;
    /* see comment below */
    if current_status='REVIEWED' then status=2;
    if current_status='CLOSED' then status=3;
    if status > .;
    drop status percent;
  run;

  /* Important step to transpose your results */
  proc transpose data=dcfs out=dcfs2;
    id current_status;
  run;

  /* Some basic data manipulation - Reviewed
     becomes equivalent to Received */
  data dcfs2;
    SENT = 0;
    RECEIVED = 0;
    CLOSED = 0;
    set dcfs2;
    if reviewed < 1 then reviewed = 0;
    received=received+reviewed;
    drop reviewed _name_ _label_;
    sent = sent + received + closed;
    received = received + closed;
  run;
  proc transpose data=dcfs2 out=dcfs3;
  run;

  proc sort;
    by descending _name_;
  run;

  /* Write to whatever file type you want.  Here we
     send it to a spreadsheet like those used in our
     custom report examples */
  proc export data=work.dcfs3
            OUTFILE= "c:\whatever\&StudyName DCFs.xls"
            DBMS=EXCEL2000 replace;
  run;
%mend BuildDCFs;

/* Example call to macro */
%BuildDCFs(ASI01);
```

APPENDIX PROGRAM EXAMPLE 3

```
/*****************************************************
* Program Name:    PharmaSUGExample3.sas
* Author:          David Johndrow,
*                  Analytical Sciences, Inc.
* Purpose:         Demonstrate how to direct SAS output
*                  to HTML files.  Modify SAS datasets
*                  to couple hyperlinks with data
*                  points.  The 2 data steps up top
*                  emulate what might have been
*                  created in the custom report
*                  examples in this presentation.
*****************************************************/

data OverallStats;
  input Site $ 1-16 Logged 17-18
        Screening 20-21 Week1 23-24 Clean 26-27;
cards;
1101 - Lagle    3  3  2  1
1345 - Lin      1  0  0  0
2020 - Hilton   1  1  0  0
All Sites       5  4  2  1
;

data PatientListing;
  input Pt $ 1-3 Site $ 5-20 Cover $ 21
        Demog $ 23 Lab $ 25 QOL $ 27;
cards;
001 1101 - Lagle    C C C C
002 1101 - Lagle    C C D D
003 1101 - Lagle    C C C C
004 1345 - Lin      C   C L
005 2020 - Hilton   C C   C
;
run;

/* This macro allows us to create web pages containing
   each sites patient completion report */
%macro PrintSites(SiteName);
  ods html body='c:\whatever\&SiteName..html';
  proc print data=patientlisting noobs;
    where site="&SiteName";
    title "Protocol XYZ";
    title2 "Patient Completion Report - Site &SiteName";
    footnote 'C=Complete    D=Data Entry    L=Logged In';
  run;
  ods html close;
%mend PrintSites;

%PrintSites(1101 - Lagle);
%PrintSites(1345 - Lin);
%PrintSites(2020 - Hilton);


/* Modify the overall stats dataset by wrapping HTML
   code around each site name, thus making hyperlinks
   to the sites web pages created above */
data OverallStats2;
  attrib Site length=$64.;
  set OverallStats;
  if site ^= 'All Sites' then
    Site = '<A HREF="' || compress(site) ||  '.html">'
           ||  compress(Site)  || '</A>';
run;

/* Generate the HTML web page of our new overal stats
   dataset */
ods html body='c:\whatever\ODSOverall.html';
proc print data=overallstats2 noobs;
  title 'Protocol XYZ - Overall Study Status';
  footnote;
run;
ods html close;
```