Extracting Data from Oracle® into SAS® Lessons Learned

Maria Y. Reiss, Wyeth Research, Collegeville, PA

ABSTRACT

In the pharmaceutical industry, it is very common to use SAS® and the Oracle Clinical™ database system together to store, process and analyze clinical data. The traditional way of using these products is to store the clinical data in an Oracle Clinical database and analyze the data using the SAS system. Before data can be analyzed with SAS, it must be extracted from Oracle and transformed into SAS views or data sets. This paper provides an overview of the various techniques available to accomplish this task. In SAS 8, SAS Institute introduced SAS® Integration Technologies, a product that allows easier integration of SAS with other applications. This paper also discusses the use of this product to integrate SAS and Oracle.

INTRODUCTION

SAS provides several techniques for extracting data from external database management systems (DBMS) and converting the data to SAS views or data sets. In fact, each recent new release of SAS has introduced an improved technique for accessing DBMS data. At Wyeth Research, the clinical programmers have, at various times, used or experimented with all of the available techniques. This paper provides an explanation of each technique and comments on our experience with each method. The DBMS databases used at Wyeth Research to store data from clinical trials are Oracle Clinical databases. Consequently, this paper will focus exclusively on the Oracle DBMS. However, the techniques described in this paper also apply to other DBMS databases. At Wyeth Research, the Oracle Clinical databases and the SAS system reside on separate Unix servers, but the principles explained in this paper also apply to other operating systems and server arrangements.

The techniques for extracting Oracle data into SAS are:

- PROC ACCESS to create access descriptors and view descriptors (introduced in SAS 6)
- SQL Procedure Pass-Through Facility (introduced in SAS 6)
- Oracle Libname Engine (introduced in SAS 7)

The SAS Integration Technologies product was introduced in SAS 8 and provides a means for accessing SAS “in languages such as C++, Java, Microsoft’s Visual Basic, and other languages that support distributed object programming” (Vodicka). Since the purpose of the Integration Technologies product is to allow seamless integration of different applications, this paper also discusses using the product to access both the SAS system and Oracle data from within an object-oriented programming environment.

In the pharmaceutical industry, updates to clinical data are tightly controlled and audited. Wyeth Research’s policy in this regard is to only allow selected individuals to update data using Oracle Clinical tools. At Wyeth Research, the SAS system is *never* used to update data in the Oracle databases. SAS is only used to extract and analyze the data. So even though it is technically possible to update Oracle tables with some of the techniques described here, this paper will not delve into this use of the techniques.

There are, of course, some prerequisites that need to be met before trying the techniques described in this paper. These prerequisites are:

- a working connection between the Oracle database and the SAS system. All necessary components such as SQL*Net or Oracle SQL Plus must be installed and working correctly.
- a license for SAS/ACCESS.
- a userid with the authority to extract Oracle data.
- a license for SAS Integration Technologies.

ACCESS DESCRIPTORS AND VIEW DESCRIPTORS

The technique of Access Descriptors and View Descriptors uses the ACCESS Procedure, the procedure for downloading DBMS data into SAS. This procedure is part of the SAS/ACCESS® product that was introduced with SAS 6.

This technique consists of three steps:

1. Link to a single Oracle table with an Access Descriptor.

The access descriptor is code that contains the structural information about the Oracle table. The access descriptor encapsulates all the information about the Oracle table.

```
/* Create Access Descriptor to the Adverse Event Table for study 123 */

libname access_descriptor_libname 'path';
/* Note: Can use libname to save the access descriptor to a permanent library */

proc access dbms=oracle;
create access_descriptor_libname.adverse.access;
user=user-name; orapw=pass-word; table=adverse; path='path-for-study-123-OC-data';
rename oracle_variable_name = sas_variable_name ; /* optional – rename variables */
format oracle_variable_name sas_format ;          /* optional – format variables */
list all;
```

The default SAS format for Oracle data of type DATE is DATETIME16. To apply a different datetime format, use the optional format statement in the PROC ACCESS code.

2. Create views containing the desired data from the Oracle table with View Descriptors.

SAS Institute defines a view descriptor as

> "a file type that uses an Access Descriptor to locate the file and determines what parts of the data to import to SAS. These View Descriptors are used in much of the same ways that SAS datasets are used. … whenever the [view descriptor] is referenced, the data is extracted at that time from the external data source that the related Access Descriptor defines." (TS-501)

The views that are created by the view descriptors are called access views or interface views.

It is possible to update the underlying Oracle data using view descriptors if the user has the proper permissions (TS-566D).

```
/* Create 2 View Descriptors for Adverse Event data */
libname view_descriptor_libname 'path';
/* Note: Can use libname to save the view descriptor to a permanent library */

proc access dbms=oracle;
/* 1st View Descriptor = all data in the Adverse Event table */
create view_descriptor_libname.all_ae.view;
select all;

/* 2nd View Descriptor = treatment emergent adverse events only */
create view_descriptor_libname.teae_ae.view;
select all ;
subset where teae_flag='Yes';
run;
```

One can also save the access view into a SAS dataset. This way, one can access the view without relying on the access descriptor.

```
proc access viewdesc=view_descriptor_libname.teae_ae
out=sas_dataset_libname.adverse_teae;
```

In SAS, data sets and views cannot have the same name, so the OUT= data set must have a different name than the access view.

3. Access the data within SAS code.

One can access the data by referring to the view descriptor:

```
proc print data=view_descriptor_libname.teae_ae width=min;
   id patient;
   title1 “Study xxx”;
   title1 “Patients with Treatment Emergent Adverse Events”;
run;
```

The libname must be specified even if it is the work library.

When Wyeth Research was using SAS 6, this was the method we used to extract Oracle data. While this method worked fairly well, the following disadvantages caused us to discontinue using it:

- This method requires the creation of two physical files (the access descriptor and the view descriptor) for every Oracle view. Since the Wyeth Research clinical database requires a large number of such views, this method is expensive in terms of storage space.
- This technique is cumbersome to use. Frederick Pratter described it as “probably the worst designed SAS product of the decade” (Pratter).

THE SQL PROCEDURE PASS-THROUGH FACILITY

The next technique for extracting Oracle data, the SQL Procedure Pass-Through Facility, is also part of the SAS/ACCESS product. The Pass-Through Facility was introduced in SAS 6.07 and enhanced in SAS 7 and 8.

To access an Oracle table with the Pass-Through Facility, code PROC SQL with five statements:

1. A CONNECT statement to connect with the Oracle database.

   ```
   connect to oracle (user=”&user” orapw=”&orapw” path=”&path”);
   ```

2. A statement to save the results of the SQL query in either a SAS view or data set.

   To save the results in a SAS view:

   ```
   create view sas_view_name
   ```

   To save the results in a SAS data set:

   ```
   create table sas_table_name
   ```

   This statement is not mandatory, but if it is omitted, the results of the SQL query are not saved. Instead the results are only displayed in the output window. Note that views created with Proc SQL cannot be modified.

3. A pass-through SQL query.

   This SQL statement selects the desired data from the Oracle table. It must be written in Oracle SQL syntax, and is passed to the Oracle database server for processing. This query can join multiple Oracle tables and it can be as complex as needed. This statement is also called the inner select statement because it is contained within another SQL statement.

4. A SAS SQL statement.

   This statement is the SELECT statement portion of the PROC SQL code. It is also called the outer select statement because it contains the pass-through SQL statement. In this query, you can perform additional formatting and/or subsetting of the data returned from the inner select statement.

5. A DISCONNECT statement to disconnect from the Oracle database.

   ```
   disconnect from oracle;
   ```

Example of the SQL Procedure Pass-Through Facility

Below is a simple example of connecting to an Oracle database. This example comes from the SAS web site at the URL http://ftp.sas.com/techsup/download/sample/unix/access/oracle/create_dset_sql.html .

```
/*******************************************************************\
| Copyright (C) 1997 by SAS Institute Inc., Cary, NC, USA.          |
|                                                                   |
| SAS (R) is a registered trademark of SAS Institute Inc.           |
|                                                                   |
| SAS Institute does not assume responsibility for the accuracy of  |
| any material presented in this file.                              |
\*******************************************************************/
/* This example shows how to create a permanant sas dataset from
   a connection to oracle. The example uses oracle as the sample
   database, but it would also work with the connection to
   sybase, informix, db2 and ingres                            */

libname libref 'SAS-data-Library';
options dbdebug;

proc sql;
     connect to oracle ( user=oracle_id orapw=oracle_passwd
path="@t:oracle_server:oracle_sid");
          create table libref.dataset_name as select * from connection to
          oracle ( select * from oracle_table );
disconnect from oracle;
quit;
```

The SAS web site contains many other examples of using the PROC SQL Pass-Through Facility. The complete list of examples can be viewed at http://ftp.sas.com/techsup/download/sample/unix/access/oracle/ .

SAS 7 saw an additional performance improvement to the Pass-Through Facility -- the implicit SQL passthru. SAS defines this method as

> “a PROC SQL feature that, for performance-sensitive SQL operations, will transparently convert your PROC SQL query into a DBMS-specific SQL query and directly pass this converted query to the DBMS for processing” (Levine).

Performance-sensitive operations are defined as queries that

> “reference a single SAS/ACCESS engine libref and contain one or more of the following:
>
> - the DISTINCT keyword
> - JOINS (inner and outer)
> - SQL aggregate functions
> - UNIONS” (Levine)

As with PROC ACCESS, the default SAS format here for Oracle data of type DATE is DATETIME16. To override this default, use the FORMAT keyword on the SQL SELECT statement. For example:

```
proc sql;
Connect to oracle … ;
Create table xxx as
Select date1 format=datetime7.
From connection to oracle (select * from yyy);
Disconnect from oracle;
Quit;
```

It is not possible to update Oracle data with an SQL Procedure pass-through query.

At Wyeth Research, we converted to the PROC SQL pass-through method from the access and view descriptor technique shortly after the Pass-Through Facility became available. In our experience, this technique has several advantages. It:

- allows pre-processing of the data within Oracle Clinical prior to conversion to SAS.
- provides the flexibility and power of the SQL language.
- allows the choice of creating an output SAS data set or view from the Oracle data.
- provides the performance advantage of the implicit SQL passthru feature.

However, this technique does have the disadvantage of requiring the use of two different forms of SQL -- Oracle and SAS -- each with a different syntax.

THE ORACLE LIBNAME ENGINE

SAS 7 introduced a new method for connecting to Oracle databases – the Oracle Libname Engine. With this method, one can assign a library reference directly to an Oracle database with a LIBNAME statement. For example:

```
libname db-libref oracle user=scott password=tiger path=oracle-server-path
        preserve_col_names=yes /*optional – to retain lowercase column names */;
```

where *oracle* is the name of the DBMS engine for Oracle.

This libname statement creates a connection to an entire database, not just an individual table. It creates a library on top of the Oracle schema for all the tables in the schema. Views for all the tables in the schema are automatically created with this technique – it is not necessary to explicitly create individual view definitions.

With this libname statement, one can access the Oracle data in the same way that one would access a SAS data set. For example:

```
proc print data=db-libref.view-name; run;
```

One can also use the libref to obtain a list of tables and views in the Oracle database:

```
proc datasets lib=db-libref;
```

Or to view the columns in any Oracle table:

```
proc contents data=db-libref.view-name; run;
```

If a user has the proper authority, he or she can use the Oracle Libname Engine to create new Oracle tables. For example:

```
data db-libref.new-table-name;
  input …;
  infile xxx;
run;
```

However, if the table *db-libref.new-table-name* already exists, this code will fail. While it is possible to update Oracle tables or create new Oracle tables with the Oracle Libname Engine, replacing existing tables is not permitted (Gona and Van Wyk).

The Oracle Libname Engine method of extracting data from an Oracle database is easy to use. It allows users to access Oracle data as SAS views and requires no knowledge of Oracle SQL. If one wants to simply copy Oracle tables into SAS, then this is the method to use.

However, this technique does not allow pre-processing of the Oracle data within Oracle prior to its conversion to SAS format. If one is interested in more complex processing, such as subsetting data or joining multiple Oracle tables into a single SAS view, then the PROC SQL Pass-Through Facility is a better choice.

SAS INTEGRATION TECHNOLOGIES INTEGRATED OBJECT MODEL

At Wyeth Research, Oracle and SAS are the cornerstone products of our clinical data management and analysis system. Our traditional way of combining these two products is to use Oracle to store and manage the clinical data, then extract the data into SAS for reporting and analysis. We continue to do this at Wyeth Research, but now also use the SAS Integration Technologies product to build composite applications combining SAS, Oracle, Java, and Delphi. This section of the paper discusses how we are using SAS Integration Technologies to expand upon our traditional uses of SAS and Oracle.

The SAS Integration Technologies product provides a means of accessing SAS functionality within languages and applications that support distributed object programming. It has three components:

- Integrated Object Model (IOM) – supports COM/DCOM and CORBA.
- Third Party Messaging software (IBM's MQSeries and Microsoft's MSMQ).
- Publishing Framework (support for LDAP)

The only component that is applicable to this paper is the Integrated Object Model. The messaging and publishing framework components are not discussed in this paper.

The SAS Integrated Object Model includes the following hierarchy of objects:

SAS Workspace

▪ **LanguageService**	- submit SAS code, retrieve log and list files, listen for events (e.g. DatastepComplete, ProcComplete, StepError, SubmitComplete).
▪ **DataService**	- create or manipulate SAS library references, access data through JDBC, ADO, or OLE/DB.
▪ **FileService**	- access and manage SAS file references and directories.
▪ **Utilities**	- access and create result packages, formats, and host information.

Programmers can access these IOM objects in their code (C++, Java, Visual Basic, etc) to submit SAS language statements to a SAS server and then get the output back from SAS. Thus, the IOM allows users to seamlessly integrate SAS with other applications.

An excellent reference for these topics as they relate to SAS Integration Technologies is the SAS white paper "SAS Integration Technologies: A Roadmap" available at http://support.sas.com/rnd/itech/papers/index.html .

At Wyeth Research, we currently use the SAS IOM in the following ways:

- in web pages to dynamically display data from Oracle databases.
- within a code library and scheduling application that adds derived data to a clinical data warehouse.
- within a Delphi/Java application that generates clinical data reports from data in an Oracle database.

At a general level, the steps to use the SAS IOM to extract and process Oracle data within an application are:

1. Define the server properties of the Oracle Clinical database server.

2. Access the SAS Workspace Manager.

The SAS Workspace Manager

"performs three functions:

- It establishes a connection with SAS, returning a SAS Workspace.
- It provides pooling of SAS Workspaces which is useful in a web environment where a SAS Workspace is used briefly and then discarded.
- It manages connections with SAS, allowing you to obtain previously created SAS Workspaces." (Jahn)

3. Use the SAS Workspace Manager to create a new SAS workspace.

The SAS workspace is "roughly equivalent to a SAS session" (Stevens and Santucci). The SAS workspace can be local or remote, and one can have multiple SAS workspaces within an application. Like a SAS session, each SAS workspace will have its own WORK library which can not be shared with other workspaces.

4. Extract data from the Oracle database.

This can be done using SAS code with one of the techniques already described in this paper.

There are two ways to submit SAS code using the IOM:

- LanguageServiceSubmit method – to submit in-line SAS code.
- StoredProcessService method – to execute a stored SAS program.

The StoredProcessService method separates the SAS code from the rest of the application, thereby providing modularity and easier maintenance of the code.

Either of these methods can be used to submit SAS code to extract the Oracle data.

5. Submit additional SAS code to the SAS server to process the data returned from the Oracle database (optional).

6. Check if the SAS code executed successfully (optional).

To check if SAS code executed successfully, the application needs to listen for the StepError event.

7. Get output from SAS code.

The IOM provides the means for accessing all of the commonly generated forms of SAS output.

For more information on how to access SAS output, how to use stored processes, or the SAS IOM in general, see the SAS Integration Technologies web site at http://support.sas.com/rnd/itech/ .

Example of Using the SAS IOM

This example extracts data from an Oracle table and processes it using SAS. The output from SAS is displayed on a web page. The example combines HTML, JSP, SAS, and Oracle. (Reiss, Holland)

```
// Import needed files.
<%@ page import="com.sas.iom.WorkspaceConnector" %>
<%@ page import="com.sas.iom.WorkspaceFactory" %>
<%@ page import="com.sas.iom.SAS.IWorkspace" %>
<%@ page import="java.util.Properties" %>
<%@ page import="com.sas.iom.SAS.ILanguageService" %>
<%@ page import="com.sas.iom.SAS.ILanguageServicePackage.CarriageControlSeqHolder" %>
<%@ page import="com.sas.iom.SAS.ILanguageServicePackage.LineTypeSeqHolder" %>
<%@ page import="com.sas.iom.SAS.IWorkspace" %>
<%@ page import="com.sas.iom.SASIOMDefs.StringSeqHolder" %>
<%@ page import="com.sas.iom.SAS.IDataService" %>
<%@ page import="com.sas.rio.MVAConnection" %>
<%@ page import="java.sql.*" %>
<%@ page import="java.io.*" %>

<%  // Start writing HTML.
    out.println("<body><html>");
    out.println("<h1 align=center>Example of Using the SAS IOM</a></h1>");

try
{  // Create class for SAS Events Listener.
   class LanguageEventsListener extends com.sas.iom.SASEvents._ILanguageEventsImplBase
   { JspWriter out;
     public LanguageEventsListener(JspWriter out) { this.out = out; }
```

```
      // implement declared methods in com.sas.iom.SASEvents.ILanguageEvents
      public void ProcStart(String procname) { /* your implementation */ }
      public void SubmitComplete(int sasrc) {
        try { out.println("<p><font color=red>SAS Submit Complete.</font></p>");
            } catch(IOException ioe){} }
      public void ProcComplete(String procname) { /* your implementation */ }
      public void DatastepStart() { /* your implementation */ }
      public void DatastepComplete() { /* your implementation */ }
      public void StepError() {
        try{ out.println("<p><font color=red>SAS Step Error Occurred –
                          Check SAS Log.</font></p>"); } catch (IOException ioe){} }
}

{  // Set properties for server.
   Properties iomServerProperties = new Properties();
   iomServerProperties.put("host","host-name");
   iomServerProperties.put("port","port-number");
   iomServerProperties.put("userName","user-name");
   iomServerProperties.put("password","pass-word");
   Properties[] serverList = {iomServerProperties};

   // Setup workspace factory.
   WorkspaceFactory wFactory = new WorkspaceFactory(serverList,null,null);
   WorkspaceConnector connector = wFactory.getWorkspaceConnector(0L);
   IWorkspace sasWorkspace = connector.getWorkspace();

   // Use workspace factory to get reference to workspace stub.
   ILanguageService sasLanguage = sasWorkspace.LanguageService();

   // Set up connection point so that program can receive events.
   String cpidString = com.sas.iom.SASEvents.ILanguageEventsHelper.id();
   int d1 = (int)Long.parseLong(cpidString.substring(4,12),16);
   short d2 = (short)Integer.parseInt(cpidString.substring(13,17),16);
   short d3 = (short)Integer.parseInt(cpidString.substring(18,22),16);
   byte[] d4 = new byte[8];

   for (int i=0;i<2;i++)
     { d4[i] =
(byte)java.lang.Short.parseShort(cpidString.substring(23+(i*2),25+(i*2)),16); }

   for (int i=0;i<6;i++) {
      d4[i+2] =
(byte)java.lang.Short.parseShort(cpidString.substring(28+(i*2),30+(i*2)),16);}
   com.sas.iom.SASIOMDefs.CP_ID cpid=new com.sas.iom.SASIOMDefs.CP_ID(d1,d2,d3,d4);
   com.sas.iom.SASIOMDefs.ConnectionPointContainer cpContainer =
   com.sas.iom.SASIOMDefs.ConnectionPointContainerHelper.narrow(sasLanguage);
   com.sas.iom.SASIOMDefs.ConnectionPointHolder cpHolder = new
com.sas.iom.SASIOMDefs.ConnectionPointHolder();
   cpContainer.FindConnectionPoint(cpid,cpHolder);
   com.sas.iom.SASIOMDefs.ConnectionPoint cp = cpHolder.value;
   LanguageEventsListener sasLanguageEventsListener = new LanguageEventsListener(out);
   org.omg.CORBA.IntHolder handleHolder = new org.omg.CORBA.IntHolder();
   cp.Advise(sasLanguageEventsListener,handleHolder);
   int handle = handleHolder.value;
   // Event listener can now receive events.

   // Extract data from Oracle.
   String col_names = "study, patient, derivation_name, status, start_time ";
```

```
    String inStr = "proc SQL;";
    inStr += "connect to oracle as pcider (path='xxxxx' user='xxxxx' pass='xxxxx');";
    inStr += "create table deriv_log as select * from connection to pcider ";
    inStr += "  (select "+col_names+" from derivations.derivation_progress_log ";
    inStr += "where data_table_type='STABLE' ";
    inStr += "order by study, derivation_name, patient, start_time desc);";
    inStr += "quit;";
    sasLanguage.Submit(inStr);

    // Process data using SAS inline code.
    inStr = "proc sort data=deriv_log nodupkey; by study derivation_name patient; run;";
    inStr += "proc tabulate data=deriv_log;";
    inStr += "  class study derivation_name status;";
    inStr += "  table study*derivation_name all, status*N;";
    inStr += "run;";
    sasLanguage.Submit(inStr);

    // Get the SAS Listing.
    CarriageControlSeqHolder lstCCHldr = new CarriageControlSeqHolder();
    LineTypeSeqHolder lstLTHldr = new LineTypeSeqHolder();
    StringSeqHolder lstHldr = new StringSeqHolder();
    sasLanguage.FlushListLines(Integer.MAX_VALUE, lstCCHldr, lstLTHldr, lstHldr);

    // Output the SAS Listing.
    String[] lstLines = lstHldr.value;
    out.println("<h3 align=center>SAS LISTING</h3>");
    out.println("<pre>");
    for (int j=0; j < lstLines.length; j++)  {
           out.println(lstLines[j]); }
    out.println("</pre>");

    // Get the SAS Log.
    CarriageControlSeqHolder logCCHldr = new CarriageControlSeqHolder();
    LineTypeSeqHolder logLTHldr = new LineTypeSeqHolder();
    StringSeqHolder logHldr = new StringSeqHolder();
    sasLanguage.FlushLogLines(Integer.MAX_VALUE, logCCHldr, logLTHldr, logHldr);

    // Output the SAS Log.
    String[] logLines = logHldr.value;
    out.println("<h3 align=center>SAS LOG</h3>");
    out.println("<pre>");
    for (int i=0; i < logLines.length; i++)  {
          out.println(logLines[i]);
        }
    out.println("</pre>");

    // Cleanup
    wFactory.shutdown();
    connector.close();
}
catch (Exception e) { }
    out.println("</body></html>");
%>
```

The SAS IOM is much more than a technique for extracting Oracle data. It is a tool that allows users to integrate the SAS system with other languages and applications. The SAS IOM allows users to access the data processing power of SAS within any language or application that supports COM/DCOM and CORBA. If one wishes to access SAS and Oracle data within a Java, C++ or Visual Basic application, the SAS IOM provides the means to do this.

CONCLUSION

At Wyeth Research, SAS and Oracle form the backbone of our clinical data management and analysis system. This paper has provided an overview of the evolution of our use of these products. Initially, we wrote stand-alone SAS programs that utilized access and view descriptors to extract data from Oracle tables. More recently we began using the SQL Procedure Pass-Through Facility and the Oracle Libname Engine. Today we are also utilizing SAS Integration Technologies to combine SAS and Oracle functionality with Java, Delphi, and HTML.

REFERENCES

Charlesworth, Tim. “OVERVIEW SAS Integration Technologies”. SAS Institute Inc. Australia (1999).

Gona, Vino and Van Wyk, Jana. “Version 7 Enhancements to SAS/ACCESS® Software”. SAS Institute Inc. Cary, NC.

Holland, Philip R. “Could Enterprise Guide Have Been Written in Java?” Holland Numerics Ltd. Royston, Herts, UK.

Ivey, William. “Using Open Access Technology to Integrate SAS and ESRI Applications”. SAS Institute Inc. Cary, NC.

Jahn, Daniel. “Developing an Open Client in Visual Basic”. SAS Institute Inc. Cary, NC.

Johndrow, David. “SAS® Techniques to Get the Most out of Oracle Clinical®”. Analytical Sciences, Inc. Durham, NC.

Levine, Fred. “Using the SAS/ACCESS® Libname Technology to Get Improvements in Performance and Optimizations in SAS/SQL® Queries”. SAS Institute Inc. Cary, NC.

Nipko, Joseph. “Using SAS Integration Technologies to Interface with Enterprise Applications Written in Visual C++”. Qualex Consulting Services. Scottsdale, AZ.

Pratter, Frederick. “Access to Relational Databases Using SAS ®”. The Pharmaceutical Industry SAS Users Group 2002 Conference. www.pharmasug.org/psug2002/bp2002/tu01.pdf

SAS Institute Inc. (1999), *SAS/ACCESS Software for Relational Databases: Reference*; pp 414-422, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1999), *SAS OnlineDOC®*, Version 8, “Introducing the SAS ODBC Driver: What Do I Need to Know about the SAS System?”, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1999), *SAS OnlineDOC®*, Version 8, “SAS Data Views”, Cary, NC: SAS Institute Inc.

SAS Institute Inc. SAS Integration Technologies Web Site: http://support.sas.com/rnd/itech/ . SAS Institute Inc. Cary, NC.

SAS Institute Inc. TS-501: Data Acquisition and Exportation in PC SAS®. SAS Institute Inc. Cary, NC.

SAS Institute Inc. TS-518D: SAS/ACCESS® Guidelines for Connecting to ORACLE® Databases in the UNIX® Environment. SAS Institute Inc. Cary, NC.

SAS Institute Inc. TS-566D: Processing ORACLE® Dates with SAS/ACCESS® in the UNIX® Environment. SAS Institute Inc. Cary, NC.

SAS Institute Inc. TS-609: Minimum Requirements for using SAS/ACCESS® Software for Relational Databases in the UNIX® Environment. SAS Institute Inc. Cary, NC.

Stevens, Jay L. and Santucci, Brian. “Integrating SAS with an Open World: Java, JSP, LDAP, and Oracle”. Whitehurst Associates, Inc and THINKologies Inc. Atlanta, GA.

The University of North Carolina at Chapel Hill. “How can we access Oracle through SAS/Access?” http://www.unc.edu/atn/statistical. The University of North Carolina at Chapel Hill. Chapel Hill, NC.

Torr, Mark and Jenisch, Steve. “SAS Integration Technologies: A Roadmap”. SAS Institute Inc. Cary, NC.

Vodicka, Scott. “Enterprise Integration Technologies – What is it and what can it do for me?” SAS Institute Inc. Cary, NC.

Williams, Valerie and Chan, Brian. “Using SAS ® IOM with a Java Client Application”. Wyeth Research. Western Users of SAS Software Group 2004 Conference. http://www.wuss.org/conference/wuss12.htm

ACKNOWLEDGEMENTS

The author would like to express her appreciation to Robert Moyer, Valerie Williams and Michael Jessup of Wyeth Research for their assistance with this paper.

CONTACT INFORMATION

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

Maria Y. Reiss
Wyeth Research
500 Arcola Road
Collegeville, PA 19426
Email: reissm@wyeth.com
Phone: (484) 865-5694