Paper A9-2007

How to Implement Security Systems Using SAS

David P. Wells, Sinclair Community College, Dayton, OH

ABSTRACT

At Sinclair Community College, we have fairly complex security needs. This includes merging information from both our Student Information System (SIS) and our Active Directory (AD) system. The merged information includes user information (i.e., their network ID, name, and school ID), their groups, and any privileged information that the user can view (i.e., their budget information).

INTRODUCTION

At Sinclair we have recently (in the last few years) implemented the Business Intelligence (BI) suite from SAS. We have a data warehouse that echoes the data in our Student Information System (SIS). The SIS is the primary information system for the college, similar to many other Enterprise Resource Planning (ERP) systems used by countless companies. Like most ERPs, it has a robust security system built into it that is role based. Our mandate by the security team was to keep the privileges in SAS consistent with our SIS.

PRE-EXISTING SECURITY

Our guideline, from the security committee, is to only give access to SAS to those individuals who are employees and have SIS accounts. The primary reason for this is that a user has to sign FERPA agreements to get a SIS account, thus has gone through the needed training to view student information.

Further, if a user is blocked from seeing information on the SIS, then they will also be blocked from seeing the same information on SAS. An example is budget information. Only budget managers can see a budget and, for that matter, can only see their budget and those under them.

Role information can be pulled from two different places, either AD or the SIS. These two different systems need to be merged and brought into SAS.

Unfortunately, the role information in the SIS can either grant or deny access to a piece of data. A deny overrides any grant; therefore, a user with two roles can be granted permissions to view data from one role, but then denied permission to the same data by the other role. Even though there is a specific grant, the deny takes priority, and the user will not be able to see the information.

The other translation needed from our SIS to SAS is the UserIDs. In our SIS, we use the UNIX UserID, which, of course, is different from the Windows NT / SAS UserID. Fortunately, there is a table in our SIS that has the mapping.

DETAILED EXAMPLE

Our SIS implements security by screen. The method of doing this is through a series of Security Class tables that contain lists of screens. Some of these screens are listed as NEVER DO, INQUIRY ONLY or DO ONLY (fields 2, 3, and 9 respectfully). For example:

Class Name	PROHIBITED	INQUIRY.ONLY	LIMITED.TO
RSR.SUPER.IT	EMER	AADT	AADT
	ECWP	ACAP	AC
	FIWP	ACDR	ACAP
		ACLV	ACDR
		ACOI	ACLV
		ACP	ACOI
		ACPR	ACP
		ACS	ACPR
		ACSI	ACS
		ACTI	ACSI

		ACWP ADAI ADFR (etc…)	ACTI ACWP ADAI ADFR (etc…)

The SIS contains a table listing the security classes the user belongs to and the user's UNIX ID. These classes map to groups in SAS. For example:

UNIX ID	SYS.USER.CLASSES	SYS.PERSON.ID
DWELLS	RSR.SUPER.IT ADM.SUPER.RUL.IT ADM.SUPER.IT BKS.SUPER.FLNK.IT BKS.SUPER.IT BUR.SUPER.IT BUR.SUPER.FLNK.IT FA.SUPER.IT FA.SUPER.FLNK.IT SENIOR.SUPER.IT XAV.SUPER.IT REPORT.ALL	0468233

Each screen listed in the first table equates to a list of fields in Colleague and to a list of fields in the data warehouse and in SAS. Once mapped to the AD UserID (via the PERSON table), we get a list of data elements available for each user that is either a grant access or deny access to.

MAPPING SECURITY ROLES

We have a job that runs every night to re-create our security mappings. This job first calls a Java program (called Extract_DW_Security.java) to pull the SIS information and place it into a MS SQL database. After the data is loaded into MS SQL, a SAS program (called ADUsers.sas) merges the existing SAS Security information with AD and SIS information.

For a user to gain access to our SAS System, they must have both a SIS UserID and a cleanly mapped AD UserID. It is important to note that the user must reside in the Users Operational Unit (OU) of AD to be cleanly mapped. We have several OUs in our AD cluster including the Students and Non-Employees which do not have access to SAS.

EXTRACT_DW_SECURITY.JAVA

As mentioned above, the Java program pulls the information from our SIS. This is broken into several steps:

- Get the list of SIS Roles
- Get the list of users by SIS Role
- Get the list of users with AD UserID

Given these three tables, we then create a table containing AD UserID and SIS Role. This table, along with the list of Roles, is used in the ADUsers.sas.

ADUSERS.SAS

This is the heart of our import of security. The listing for this routine is in Appendix I, and the modified **importad.sas** routine is in Appendix II.

CURRENT USAGE

Today, we have a robust security with roles (from both our SIS and AD) and have the ability to link applications back to these roles. This metadata is updated on a nightly basis and; when people arrive, move or leave, the security is updated.

For example, if a user moves from the Finance department to the Facilities department, they would lose access to budget data in our SIS. That night, they would lose the same access in SAS for any reports or data that are marked

for that role (or group).

FUTURE ENHANCEMENTS

The security on our SIS is based on screens and, for the most part, these map to tables in SAS. There are a few exceptions that we don't handle quite correctly at the moment. These are cases where we would like to limit data at the field level instead of the table level.

An example of this would be the Social Security Number for a student. We would like to be able to have the access to this field to be automated for the few departments that need to see the information (i.e., only users in HR, Registration, or the Bursar's Office.) With our system, the only way to do this is manually setting security on the field.

CONCLUSION

Combining the existing metadata from our SIS to that in AD, we have been able to implement a security system in SAS that keeps track of the changing environment of our staff and faculty.

ACKNOWLEDGMENTS

I would like to thank SAS Consulting for setting up the original security setup that provides the basis for the existing system.

CONTACT INFORMATION (HEADER 1)

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

David P. Wells
Sinclair Community College
444 West Third St.
Dayton, OH 45402

Work Phone:	937-512-2454
Fax:	937-512-2049
E-mail:	david.wells@sinclair.edu
Web:	www.sinclair.edu

APPENDIX I ADUSER.SAS

The following code is, with bolded modifications for security reasons, our **ADUser.sas** code:

```
OPTIONS MPRINT SYMBOLGEN;

/*   Incremental Load of Sinclair Employees  */

LIBNAME DWMeta OLEDB  PROPERTIES=(
"Initial Catalog"=DATABASE)  PROVIDER=SQLOLEDB.1
DATASOURCE='SERVER'  PROMPT=NO  USER=USERID
PASSWORD="PASSWORD" ;

%macro updomr(delper=,grp=);
%put &delper;
%if &delper < 2000 %then %do;
        filename out1 'D:\temp\Update&grp...xml';
        filename out2 'D:\temp\Update&grp..xml';
        %mduchgl(change=tmpdata2,temp=work,outrequest=out1,outresponse=out2);
%end;
%else %do;
data _null_;
        abort abend 100;
run;
%end;
%mend;

%include 'SASPATH\SAS\SAS 9.1\core\sample\importad.sas';

%mduextr(libref=work);

data filters;
    format tablename filter $100.;
        tablename = 'IDGRPS';
        filter = 'upcase(name) = "ADMINISTRATORS"';
run;
data filters2(keep=tablename filter);
    format tablename filter $100.;
    set logins;
        if externalkey = 0;
        tablename = "LOGINS";
        FILTER = 'upcase(userid) = "' || trim(upcase(userid)) || '"';
RUN;
PROC APPEND BASE=FILTERS DATA=FILTERS2;

proc sql noprint;
    create table tempgrp as
        select distinct class_nm from dwmeta.v_security_person_attr_access;

        create table tmpgrpmem as
        select distinct person_id_nb, class_nm from dwmeta.v_security_person_attr_access;
        data idgrptbla ;
            %defineidgrpcols;
                set tempgrp;
                keyid = class_nm;
                description = class_nm;
```

```
                        name = class_nm;
                        drop class_nm;
        run;
        proc append base=tempdata.idgrps data=idgrptbla;
        data idgrpmemtbla;
          %defineidgrpmemscols;
                        set tmpgrpmem;
                        grpkeyid = class_nm;
                        memkeyid = put(input(person_id_nb,9.),z9.);
                        drop person_id_nb class_nm;
        run;
        proc append base=tempdata.grpmems data=idgrpmemtbla;

%mducmp(master=tempdata, target=work, change=tmpdata2, externonly=1,exceptions=work.filters);

data tmpdata2.person_summary;
        set tmpdata2.person_summary;
        where name not in ('karl.konsdorf','david.wells');
run;
data tmpdata2.person_add;
        set tmpdata2.person_add;
        if name not in ('karl.konsdorf','david.wells');
run;

%mduchgv( target=work, change=tmpdata2, temp=tempdata, errorsds=tempdata.errors);

proc sql noprint;
    select count(*) into :delper from tmpdata2.person_delete;
quit;


%updomr(delper=&delper,grp=emp);
```

APPENDIX II IMPORTAD.SAS

The following code is the modified version of **importad.sas** from the code samples provided by SAS. Note the bolded text has been changed:

```
/*****************************************************************************
 *          S A S   S A M P L E   L I B R A R Y
 *
 *      NAME: IMPORTAD
 *     TITLE: Metadata User Import From Active Directory
 *   PRODUCT: SAS
 *   VERSION: 9.1
 *    SYSTEM: ALL
 *      DATE: 03DEC2003
 *      DESC: Example code to extract user information from an Active
 *            Directory and load it into the Metadata Server.
 *      KEYS: METADATA USER PERSON IDENTITYGROUP GROUP LOGIN
 *
 *****************************************************************************/
 LIBNAME tempdata 'D:\temp\data';
 LIBNAME tmpdata2 'D:\temp\data2';
/*****************************************************************************
 *****************************************************************************
 **                                                                         **
 **  The following SAS Program is divided into 5 discrete sections in       **
 **  order to help simplify its overall organization.  Each SECTION is      **
 **  marked by a comment box like this one with "double bound" asterisk.    **
 **  Here is a summary of the sections:                                     **
 **                                                                         **
 **  SECTION 1: SAS Option, Macro Variable, and filename Definitions        **
 **                                                                         **
 **  SECTION 2: %mduimpc defines canonical datasets and variable lists.     **
 **                                                                         **
 **  SECTION 3: Extract User Information from Active Directory, normalize   **
 **             data, and create corresponding canonical datasets.          **
 **                                                                         **
 **  SECTION 4: Extract Group Information from Active Directory, normalize  **
 **             data, and create corresponding canonical datasets.          **
 **                                                                         **
 **  SECTION 5: %mduimpl reads the canonical datasets, generates            **
 **             XML representing metadata objects, and invokes PROC         **
 **             METADATA to load the metadata.                              **
 **                                                                         **
 **  In order to run this program, you will modify the connection parms     **
 **  for the Active Directory Directory Server where user information is    **
 **  read from and the SAS Metadata Server that recieves this information   **
 **  in the form of XML representing metadata objects.  These connection    **
 **  parms are found in SECTION 1 below.                                    **
 **                                                                         **
 **  CAUTION: before running this program, please read the SAS code below,  **
 **  SECTION by SECTION to gain an understanding of its overall flow.  It   **
 **  is especially important to understand the ldap filters used to         **
 **  retrieve persons in SECTION 3.  Depending on the volume of defined     **
 **  users, your user selection criteria, and local site restrictions,      **
 **  the filters may require little or much modification.  Also note that   **
 **  the same principles apply for the ldap filters used to retrieve        **
 **  groups in SECTION 4.                                                   **
 **                                                                         **
 **  NOTE: if a macro is defined with the name "_EXTRACTONLY", then         **
 **  no attempt will be made to load the user information into the          **
```

```
 **  metadata server.  Only the extraction of the user information from    **
 **  Active Directory and the creation of the canonical datasets. This     **
 **  will be used with synchronization processing.                         **
 **                                                                        **
 ************************************************************************
 ************************************************************************/

%LET _EXTRACTONLY = YES;

/************************************************************************
 ************************************************************************
 **                                                                      **
 **  SECTION 1: SAS Option, Macro Variable, and filename Definitions     **
 **                                                                      **
 ************************************************************************
 ************************************************************************/

/************************************************************************/
/* Use the Meta* options to specify the metadata server connection options  */
/* where the user information will be loaded.                               */
/************************************************************************/
options metaserver="YOUR SAS METADATA SERVER" /* network name/address of the      */
                                     /*   metadata server.              */

        metaport=8561               /* Port Metadata Server is listening on.*/

        metauser="USERID"           /* Domain Qualified Userid for          */
                                    /*   connection to metadata server.     */

        metapass="PASSWORD"         /* Password for userid above.           */

        metaprotocol=bridge         /* Protocol for Metadata Server.        */

        metarepository=foundation;  /* Default location of user information */
                                    /*   is in the foundation repository.   */


/************************************************************************/
/* The following macro variables supply connection parameters for the       */
/* Active Directory (AD) Server containing the user and group information.  */
/************************************************************************/

%let ADServer = "YOUR AD SERVER"; /* network name/address of   */
                                        /*   the AD server.          */

%let ADPort   = 389;                    /* Port on which LDAP interface*/
                                        /* is listening. 389 is a      */
                                        /* standard port.              */

%let ADPerBaseDN ="cn=Users,dc=scc-nt,dc=sinclair,dc=edu"; /* Specify the
Distinguished */
                                        /* Name in the LDAP hierarchy  */
                                        /* where People searches begin.*/

%let ADGrpBaseDN ="cn=Users,dc=scc-nt,dc=sinclair,dc=edu"; /* Specify the
Distinguished */
                                        /* Name in the LDAP hierarchy  */
                                        /* where Group searches begin. */


%let ADBindUser = "USERID";      /* Userid which will connect   */
```

```
                                        /* to the AD server and extract*/
                                        /* the information.            */

%let ADBindPW = "PASSWORD";             /* Password for Userid above.  */


/*****************************************************************************/
/* Define the tag that will be included in the Context attribute of          */
/* ExternalIdentity objects associated with the information loaded by this   */
/* application.  This tag will make it easier to determine where information*/
/* originated from when synchronization tools become available.              */
/* Note, the value of this macro should not be quoted.                       */
/*****************************************************************************/
%let ADExtIDTag = Active Directory Import;

/*****************************************************************************/
/* This process will extract the ActiveDirectory information into datasets   */
/* in the libref represented by the "extractlibref" macro variable.  The     */
/* extracted information will be cleansed and normalized in these datasets   */
/* and then transferred into the canonical form datasets defined in the      */
/* %mduimpc macro.                                                           */
/*                                                                           */
/* Specify the library to where the ActiveDirectory information should       */
/* be extracted.                                                             */
/*****************************************************************************/
%let extractlibref=tempdata;


/*****************************************************************************/
/* Choose the value that will be used as the keyid for Person information.   */
/* Choices are the DistinguishedName of the User entry or the employeeid.    */
/* For groups, the DistinguishedName will be used.                           */
/*                                                                           */
/* %let keyidvar=employeeID;                                                 */
/* %let keyidvar=distinguishedName;                                          */
/*****************************************************************************/
%let keyidvar=employeeID;


/*****************************************************************************/
/* Set the name of the AuthenticationDomain in the metadata to which logins */
/* created by the process should be associated.  Note, this name is not      */
/* required to be the same name as the windows domain. Logins from multiple */
/* windows domains can participate in the same metadata AuthenticationDomain*/
/* if the windows domains trust each other.                                  */
/*****************************************************************************/
%let MetadataAuthDomain=DefaultAuth;


/*****************************************************************************/
/* Set the name of the windows domain that should be prepended with a '\'    */
/* to each login created by this extraction.                                 */
/*****************************************************************************/
%let WindowsDomain=YOUR DOMAIN;      /* ours is SCC-NT; */

/*****************************************************************************/
/* The importlibref macro variable declares the libref where the normalized */
/* datasets defined by the macro %mduimpc will be created in the processing */
/* below. It is VERY important to NOT change any &importlibref reference in */
/* the code below.  If you want to save the normalized datasets in a         */
/* specific library then uncomment libname xxxx 'your_path_name';.           */
```

```
/* supply your own path name, and change  %let importlibref=work; to        */
/* %let importlibref=xxxx; where xxxx is a libref name of your choosing.    */
/***************************************************************************/

/* libname xxxx 'your_path_name'; */
%let importlibref=tempdata;

/***************************************************************************/
/* filename for location where macro %mduimpl saves its generated XML.      */
/* This can be a fully qualified filename including the path and .xml       */
/* extension.                                                               */
/***************************************************************************/
filename keepxml "request.xml" lrecl=1024;



/***************************************************************************
 ***************************************************************************
 **                                                                       **
 **  SECTION 2: %mduimpc defines canonical datasets and variable lists.   **
 **                                                                       **
 ***************************************************************************
 ***************************************************************************/

/***************************************************************************/
/* Invoke the %mduimpc macro to generate the macro variables used           */
/* to define the canonical datasets and columns for input to the %mduimpl   */
/* macro. The %mduimpl (see SECTION 5: at end of program) macros            */
/* reads the canonical form datasets, builds an XML stream containing       */
/* user information, and loads this user information into the metadata      */
/* server specified in the meta options above.                              */
/***************************************************************************/
%mduimpc(libref=&importlibref,maketable=0);



/***************************************************************************
 ***************************************************************************
 **                                                                       **
 **  SECTION 3: Extract User Information from Active Directory, normalize **
 **             data, and create corresponding canonical datasets.        **
 **                                                                       **
 ***************************************************************************
 ***************************************************************************/

/***************************************************************************/
/* MACRO: ldapextrpersons                                                   */
/*                                                                          */
/* To extract user information from ActiveDirectory (AD), the LDAP datastep */
/* interface is used to connect and query AD.                               */
/*                                                                          */
/* This macro is used within a datastep which has established an ldap       */
/* connection.  Because some servers will limit the number of directory     */
/* entries retrieved on a single search, the datastep will be built with a  */
/* series of filters that are used in this macro to select the entries that */
/* will be processed by the macro.                                          */
/*                                                                          */
/* AD ships with standard schemas that define much of the information       */
/* needed here. However, the standard schema is often extended with         */
/* additional site-specific attributes.  If your site has extended the      */
/* scehma, you will need to obtain this information from your local Active  */
```

```
/* Directory administrator and modify the ldapextrpersons macro accordingly.*/
/*************************************************************************/

%macro ldapextrpersons;
       shandle=0;
       num=0;

       /* The attrs datastep variable contains a list of the ldap attribute */
       /* names from the standard schema. */
       attrs="displayName streetAddress cn company mail employeeID " ||
             "facsimileTelephoneNumber distinguishedName l "         ||
             "mobile otherTelephone physicalDeliveryOfficeName "     ||
             "postalCode name sAMAccountName st "                    ||
             "telephoneNumber co title whenChanged whenCreated";

       /*****************************************************************/
       /* Call the SAS interface to search the LDAP directory.  Upon    */
       /* successful return, the shandle variable will contain a search */
       /* handle that identifies the list of entries returned in the    */
       /* search.  The num variable will contain the total number of    */
       /* result entries found during the search.                       */
       /*****************************************************************/
       call ldaps_search( handle, shandle, filter, attrs, num, rc );
       if rc NE 0 then do;
         msg = sysmsg();
         put msg;
         put filter=;
       end;

       do eIndex = 1 to num;
          numAttrs=0;
          entryname='';

          call ldaps_entry( shandle, eIndex, entryname, numAttrs, rc );
          if rc NE 0 then do;
             msg = sysmsg();
             put msg;
          end;

          /* initialize the entry variables */
          displayName="";
          streetAddress="";
          cn="";      /* common name */
          company="";
          mail="";    /* email address */
          employeeID="";
          facsimileTelephoneNumber="";
          distinguishedName="";
          l="";       /* location - city */
          mobile="";  /* mobile phone */
          otherTelephone="";
          physicalDeliveryOfficeName="";
          postalCode="";
          name="";
          sAMAccountName="";
          st="";      /* state */
          telephoneNumber="";
          co="";      /* country */
          title="";   /* job title */
          whenChanged="";
          whenCreated="";
```

```
/* for each attribute, retrieve name and values */
if (numAttrs > 0) then do aIndex = 1 to numAttrs;

   attrName='';
   numValues=0;

   call ldaps_attrName(shandle, eIndex, aIndex, attrName, numValues, rc);
   if rc NE 0 then do;
      put aIndex=;
      msg = sysmsg();
      put msg;
   end;

   /* get the 1st value of the attribute. */
   call ldaps_attrValue(shandle, eIndex, aIndex, 1, value, rc);
   if rc NE 0 then do;
      msg = sysmsg();
      put msg;
   end;

   /*********************************************************************/
   /* All of the following attrName values are MS Base Schema Supplied */
   /*********************************************************************/

   /* extract the displayName - Display-Name in */
   if (attrName = 'displayName')  then
      displayName= value;
   /* extract the streetAddress - Address */
   else if (attrName = 'streetAddress')  then
      streetAddress= value;
   /* extract the cn - Common-Name */
   else if (attrName = 'cn')  then
      cn= value;
   /* extract the Company - Company */
   else if (attrName = 'company')  then
      company= value;
   /* extract the l - Locality-Name contains city/town */
   else if (attrName = 'l')  then
      l= value;
   /* extract the mail - Email-Addresses (multi-valued) */
   else if (attrName = 'mail')  then
      mail= value;
   /* extract the employeeID - Employee-ID */
   /***********************************************/
   /* employeeid may need to be normalized/cleansed */
   /***********************************************/

   else if (attrName = 'employeeID')  then  do;
      employeeID= compress(value, "<>""");
   end;
   /* extract the facsimileTelephoneNumber - Facsimile-Telephone-Number */
   else if (attrName = 'facsimileTelephoneNumber')  then
      facsimileTelephoneNumber= value;
   /* extract the distinguishedName - Obj-Dist-Name */
   else if (attrName = 'distinguishedName')  then
      distinguishedName= value;
   /* extract the mobile - Phone-Mobile-Primary */
   else if (attrName = 'mobile')  then
      mobile= value;
   /* extract the otherTelephone - Phone-Office-Other  */
```

```
            else if (attrName = 'otherTelephone')  then
               otherTelephone= value;
            /* extract the physicalDeliveryOfficeName  */
            else if (attrName = 'physicalDeliveryOfficeName')  then
               physicalDeliveryOfficeName= value;
            /* extract the postalCode - Postal-Code */
            else if (attrName = 'postalCode')  then
               postalCode= value;
            /* extract the name - RDN  (relative distinguished name) */
            else if (attrName = 'name')  then
               name= value;
            /* **extract the sAMAccountName - SAM-Account-Name */
            else if (attrName = 'sAMAccountName')  then
               sAMAccountName= value;
            /* extract the st - State-Or-Province-Name */
            else if (attrName = 'st')  then
               st= value;
            /* extract the telephoneNumber - Telephone-Number  */
            else if (attrName = 'telephoneNumber')  then
               telephoneNumber= value;
            /* **extract the co - Text-Country */
            else if (attrName = 'co')  then
               co= value;
            /* extract the title - Title */
            else if (attrName = 'title')  then
               title= value;
            /* extract the whenChanged - When-Changed */
            else if (attrName = 'whenChanged')  then
               whenChanged= value;
            /* extract the whenCreated - When-Created */
            else if (attrName = 'whenCreated')  then
               whenCreated= value;

         end;  /* end of attribute loop */

         /*********************************************************************/
         /* If we have an employeeID then we have a person, save the info.  */
         /* If not, then move on to the next entry.  This check may need to */
         /* modified if your company uses another field as a unique         */
         /* identifier for employees/people.                                */
         /*********************************************************************/
         if employeeID NE "" then
            output &extractlibref..ldapusers; /* output to ldapusers dataset */

      end;  /* end of entry loop */

      /* free search resources */
      call ldaps_free(shandle,rc);
      if rc NE 0 then do;
         msg = sysmsg();
         put msg;
      end;

%mend;


/*******************************************************************************
**********/
/* The following datastep extracts user information from an AD using the ldap
datastep       */
/* call interface and the %ldapextrpersons macro defined above.
```

```
*/
/*
*/
/* Because some AD servers will limit the number of directory entries retrieved on a
single  */
/* search, this datastep is built with a series of filters.  Each setting of the
variable    */
/* 'filter' below is used in the %ldapextrpersons macro invocation that follows it.
*/
/*
*/
/* You may freely modify the filter= values according to restrictions imposed at
your site   */
/* and the number/selection criteria of users being imported.  Just make sure that
each      */
/* filter='your_filter_value" string is follwed immediately by %ldapextrpersons.
*/
/*************************************************************************************
**********/

data &extractlibref..ldapusers
     (keep= displayName streetAddress cn company mail employeeID
facsimileTelephoneNumber
            distinguishedName l mobile otherTelephone physicalDeliveryOfficeName
postalCode name
            sAMAccountName st telephoneNumber co title whenChanged whenCreated);

     length entryname $200 attrName $100 value $600 filter $100
            displayName $70 streetAddress $100 cn $40 company $50 mail $50
            employeeID $30 facsimileTelephoneNumber $50 distinguishedName $200
            l $50 mobile $50 otherTelephone $50 physicalDeliveryOfficeName $50
            postalCode $20 name $60 sAMAccountName $20 st $20 telephoneNumber $50
            co $50 title $50 whenChanged $30 whenCreated $30;

     handle = 0;
     rc     = 0;
     option = "OPT_REFERRALS_ON";

     /* open connection to LDAP server */
     call ldaps_open( handle, &ADServer, &ADPort, &ADPerBaseDN, &ADBindUser,
&ADBindPW, rc, option );
     if rc NE 0 then do;
        msg = sysmsg();
        put msg;
     end;

     timeLimit=0;
     sizeLimit=0;
     base='';  /* use default set at _open time */
     referral = "OPT_REFERRALS_ON";
     restart = ""; /* use default set at _open time */

     call ldaps_setOptions(handle, timeLimit, sizeLimit, base, referral, restart,
rc);

     filter="(&(&(displayName>=A)(displayName<=C))  (employeeID=*)  )";
     %ldapextrpersons

     filter="(&(&(displayName>=C)(displayName<=D))  (employeeID=*) )";
     %ldapextrpersons

     filter="(&(&(displayName>=D)(displayName<=E))  (employeeID=*)  )";
```

```
     %ldapextrpersons

     filter="(&(&(displayName>=E)(displayName<=G))  (employeeID=*)  )";
     %ldapextrpersons

     filter="(&(&(displayName>=G)(displayName<=I))  (employeeID=*)  )";
     %ldapextrpersons

     filter="(&(&(displayName>=I)(displayName<=K))  (employeeID=*)  )";
     %ldapextrpersons

     filter="(&(&(displayName>=K)(displayName<=M))  (employeeID=*) )";
     %ldapextrpersons

     filter="(&(&(displayName>=M)(displayName<=O))  (employeeID=*)  )";
     %ldapextrpersons

     filter="(&(&(displayName>=O)(displayName<=Q))  (employeeID=*)  )";
     %ldapextrpersons

     filter="(&(&(displayName>=Q)(displayName<=S))  (employeeID=*)  )";
     %ldapextrpersons

     filter="(&(&(displayName>=S)(displayName<=T))  (employeeID=*)  )";
     %ldapextrpersons

     filter="(&(&(displayName>=T)(displayName<=U))  (employeeID=*)  )";
     %ldapextrpersons

     filter="(&(&(displayName>=U)(displayName<=W))  (employeeID=*)  )";
     %ldapextrpersons

     filter="(&(&(displayName>=W)(displayName<=Y))  (employeeID=*)  )";
     %ldapextrpersons

     filter="(& (displayName>=Y)  (employeeID=*) )";
     %ldapextrpersons

     /* close connection to LDAP server */
     call ldaps_close(handle,rc);
     if rc NE 0 then do;
        msg = sysmsg();
        put msg;
     end;
run;


/*************************************************************************
*******/
/* The following datastep creates the normalized tables for person, location,
*/
/* phone, email, and login from the &extractlibref..ldapusers extracted above.
*/
/*************************************************************************
*******/

data &persontbla                      /* Macros to define Normalized Tables from
%mduimpc */
     &locationtbla
     &phonetbla
     &emailtbla
```

```
     &logintbla
      ;
     %definepersoncols;        /* Macros to define Normalized Table Columns from
%mduimpc */
     %definelocationcols;
     %definephonecols;
     %defineemailcols;
     %definelogincols;

     set &extractlibref..ldapusers;
     keyid = &keyidvar;

     /*  name is in the input ldapusers dataset already */
     /* title is in the input ldapusers dataset already */
     description=displayName;
     output &persontbl;

     /* setup location values */
     if streetAddress NE "" then do;
        locationName = name || "Office";
        locationtype = "Office";

        /* Replace carriage control chars with spaces.          */
        /* Also, the last line contains city, state, zip info,  */
        /* we already have that in the object so drop the line. */
        address = strip(translate(streetAddress,'  ','0D0A'x));

        city = l;
        /* extract data already has postal code */
        area = st ;
        country = co ;
       output &locationtbl;
     end;

     if mail NE "" then do;
        emailAddr = mail;
        emailType = "Office";
        output &emailtbl;
     end;

     if telephoneNumber NE "" then do;
        phonenumber =  telephoneNumber;
        phonetype = "Office";
        output &phonetbl;
     end;

     if facsimileTelephoneNumber NE "" then do;
        phonenumber = facsimileTelephoneNumber;
        phonetype = "Office Fax";
        output &phonetbl;
     end;

     if mobile NE "" then do;
        phonenumber = mobile;
        phonetype = "Mobile";
        output &phonetbl;
     end;

     if otherTelephone NE "" then do;
        phonenumber = otherTelephone;
        phonetype = "Other";
```

```
        output &phonetbl;
     end;

     if sAMAccountName NE "" then do;

        /* setup login values */
        /* we need to prefix the login user id with the domain id */

        if "&WindowsDomain" = "" then
           userid = sAMAccountName ;
        else
           userid = "&WindowsDomain\" || sAMAccountName ;

        password ="";
        authdomkeyid = 'domkey' || compress(upcase("&MetadataAuthDomain"));

        output &logintbl;
     end;
run;


/*************************************************************************/
/* The following datastep creates the normalized table for the           */
/* AuthenticationDomain specified in the &MetadataAuthDomain near the    */
/* beginning of this SAS code.  This value is also used to create the    */
/* foreign key variable "authdomkeyid" for the logins table in the next */
/* datastep, forming the relation between the authdomtbl and logintbl.   */
/*************************************************************************/

data &authdomtbl;
     authDomName="&MetadataAuthDomain";
     keyid='domkey' || compress(upcase("&MetadataAuthDomain"));
run;


/*************************************************************************/
/* Each person entry in &persontbl must be unique according to the       */
/* rules for Metadata Authorization Identities.  By enforcing this       */
/* uniqueness here, we help ensure that the Metadata XML will load       */
/* correctly when the %mduimpl macro is invoked with submit=1 below.     */
/*************************************************************************/

proc sort data=&persontbl nodupkey;
     by keyid;
run;

proc datasets library=&importlibref memtype=data;   /* Create Index for */
     modify person;                                 /* speedy retrieval */
     index create keyid;
run;


/*************************************************************************/
/* The location dataset should have an entry for each location that      */
/* a person will have.  So, if there are 3 people and one of them        */
/* has 2 locations, then there should be 4 records in &locationtbl.      */
/* Sort &locationtbl by the location keyid.                              */
/*************************************************************************/

proc sort data=&locationtbl nodupkey;
     by keyid;
```

```
run;


proc datasets library=&importlibref memtype=data;   /* Create Index for */
     modify location;                               /* speedy retrieval */
     index create keyid;
run;


/*************************************************************************/
/* Each person can have one or more entries in &phonetbl.  Each          */
/* entry will be a unique combination of keyid and phone number.         */
/*************************************************************************/
proc sort data=&phonetbl nodupkey;
     by keyid phonenumber;
run;

proc datasets library=&importlibref memtype=data;   /* Create Index for */
     modify phone;                                  /* speedy retrieval */
     index create keyid;
run;


/*************************************************************************/
/* Each person can have one or more entries in &emailtbl. Because        */
/* more than one person can share an EMAIL address, the entries          */
/* are not required to be unique.                                        */
/*************************************************************************/

proc sort data=&emailtbl;
     by keyid;
run;

proc datasets library=&importlibref memtype=data;   /* Create Index for */
     modify email;                                  /* speedy retrieval */
     index create keyid;
run;


/*************************************************************************/
/* Because each person can have multiple logins, the entries by          */
/* keyid are not required to be unique.  However, the UserID             */
/* attribute by relation to AuthenticationDomain must be unique for      */
/* each login owned by a person, *and* a login can only be related       */
/* to one person.  These contstraints are enforced during processing     */
/* in the %mduimpl macro, which is invoked below.                        */
/*************************************************************************/
proc sort data=&logintbl;
     by keyid;
run;

proc datasets library=&importlibref memtype=data;   /* Create Index for */
     modify logins;                                 /* speedy retrieval */
     index create keyid;
run;



/*************************************************************************
 *************************************************************************
 **                                                                     **
```

```
 **  SECTION 4: Extract Group Information from Active Directory, normalize **
 **             data, and create corresponding canonical datasets.         **
 **                                                                        **
 ****************************************************************************
 ****************************************************************************/

/****************************************************************************/
/* MACRO: ldapextrgroups                                                    */
/*                                                                          */
/* To extract group information from ActiveDirectory (AD), the LDAP         */
/* datastep interface is used to connect and query AD.                      */
/*                                                                          */
/* This macro is used within a datastep which has established an ldap       */
/* connection.  Because some servers will limit the number of directory     */
/* entries retrieved on a single search, the datastep will be built with a  */
/* series of filters that are used in this macro to select the entries that */
/* will be processed by the macro.                                          */
/*                                                                          */
/* AD ships with standard schemas that define much of the information       */
/* needed here. However, the standard schema is often extended with         */
/* additional site-specific attributes.  If your site has extended the      */
/* scehma, you will need to obtain this information from your local Active  */
/* Directory administrator and modify the ldapextrgroups macro accordingly. */
/****************************************************************************/

%macro ldapextrgroups;

       shandle=0;
       num=0;

       attrs="name description groupType distinguishedName " ||
             "sAMAccountName member whenChanged whenCreated";

       /*****************************************************************/
       /* Call the SAS interface to search the LDAP directory.  Upon    */
       /* successful return, the shandle variable will contain a search */
       /* handle that identifies the list of entries returned in the    */
       /* search.  The num variable will contain the total number of    */
       /* result entries found during the search.                       */
       /*****************************************************************/
       call ldaps_search(handle,shandle,filter, attrs, num, rc);
       if rc NE 0 then do;
          msg = sysmsg();
          put msg;
          put filter=;
       end;

       do eIndex = 1 to num;

          numAttrs=0;
          entryname='';

          call ldaps_entry(shandle, eIndex, entryname, numAttrs, rc);
          if rc NE 0 then do;
             msg = sysmsg();
             put msg;
          end;

          /* initialize the entry variables */
          name="";
          description="";
```

```
groupType="";
distinguishedName="";
sAMAccountName="";
member=""; /* DN of the group members */
whenChanged="";
whenCreated="";

/*********************************************************************/
/* for each attribute, retrieve name and values                      */
/* initialize the member attribute index to 0.  It will get set in the */
/* loop below and then used to retrieve group members after the group  */
/* attributes are set.                                               */
/*********************************************************************/
memberindex = 0;

if (numAttrs > 0) then do aIndex = 1 to numAttrs;

   attrName='';
   numValues=0;

   call ldaps_attrName(shandle, eIndex, aIndex, attrName, numValues, rc);
   if rc NE 0 then do;
      put aIndex=;
      msg = sysmsg();
      put msg;
   end;

   /*********************************************************************/
   /* if the attrName is member, then lets remember the aIndex so that */
   /* we can loop thru all the members after the group attributes are  */
   /* retrieved.                                                        */
   /*********************************************************************/
   if (attrName = 'member') then
      memberindex = aIndex;
   else do;  /* get the 1st value of the attribute. */
      call ldaps_attrValue(shandle, eIndex, aIndex, 1, value, rc);
      if rc NE 0 then do;
         msg = sysmsg();
         put msg;
      end;
   end;

   /* extract the description - Description */
   if (attrName = 'description')  then
      description=value;
   /* extract the name - RDN  (relative distinguished name)  */
   if (attrName = 'name')  then
      name=value;
   /* extract the groupType - Group-Type   */
   if (attrName = 'groupType')  then
      groupType=value;
   /* extract the distinguishedName - Obj-Dist-Name */
   if (attrName = 'distinguishedName')  then
       distinguishedName=value;
   /* **extract the sAMAccountName - SAM-Account-Name */
   if (attrName = 'sAMAccountName')  then
      sAMAccountName=value;

   /* extract the member - Member for Group */
   if (attrName = 'member')  then do;
      /* extract all the members of the group */
```

```
               member=value; /* DN of the group members */
            end;

            /* extract the whenChanged - When-Changed */
            if (attrName = 'whenChanged')  then
               whenChanged=value;
            /* extract the whenCreated - When-Created */
            if (attrName = 'whenCreated')  then
               whenCreated=value;

         end;  /* end of attributes loop */

         /* ... Group defined with no members */
         if memberindex = 0 then do;
            member="";
            output &extractlibref..ldapgrps;  /* Write out Group Name Entry */
         end;

         /* ... when Group has members then retrieve each one */
         else do;

            attrName='';
            numValues=0;

            call ldaps_attrName(shandle, eIndex, memberindex, attrName, numValues,
rc);
            if rc NE 0 then do;
               put aIndex=;
               msg = sysmsg();
               put msg;
            end;

            do i = 1 to numValues;         /* get all the members of this group. */

               call ldaps_attrValue(shandle, eIndex, memberindex, i, value, rc);
               if rc NE 0 then do;
                  msg = sysmsg();
                  put msg;
               end;
               member = value;
               output &extractlibref..ldapgrps;  /* Write out Group Member Entry */
            end;

         end;  /* end of members loop */

      end;  /* end of entry loop */

      /* free search resources */
      call ldaps_free(shandle,rc);
      if rc NE 0 then do;
         msg = sysmsg();
         put msg;
      end;
%mend;


/*************************************************************************
*********/
/* The following datastep extracts group information from an AD using the ldap
datastep     */
/* call interface and the %ldapextrpersons macro defined above.
```

```
*/
/*
*/
/* Because some AD servers will limit the number of directory entries retrieved on a
single */
/* search, this datastep is built with a series of filters.  Each setting of the
variable   */
/* 'filter' below is used in the %ldapextrpersons macro invocation that follows it.
*/
/*
*/
/* You may freely modify the filter= values according to restrictions imposed at
your site  */
/* and the number/selection criteria of groups being imported.  Just make sure that
each    */
/* filter='your_filter_value" string is follwed immediately by %ldapextrpersons.
*/
/*************************************************************************************
*********/

data &extractlibref..ldapgrps
     (keep= name description groupType distinguishedName
            sAMAccountName member whenChanged whenCreated );

     length entryname $200 attrName $100 value $600 filter $100
            name $60 description $100 groupType $20
            distinguishedName $200 sAMAccountName $20 member $200
            whenChanged $30 whenCreated $30;

     handle = 0;
     rc     = 0;
     option = "OPT_REFERRALS_ON";

     /* open connection to LDAP server */
     call ldaps_open( handle, &ADServer, &ADPort, &ADGrpBaseDN, &ADBindUser,
&ADBindPW, rc, option );
     if rc NE 0 then do;
        msg = sysmsg();
        put msg;
     end;

     timeLimit=0;
     sizeLimit=0;
     base='';  /* use default set at _open time */
     referral = "OPT_REFERRALS_ON";
     restart = ""; /* use default set at _open time */

     call ldaps_setOptions(handle, timeLimit, sizeLimit, base, referral, restart,
rc);


     filter="(&(name<=H)(objectClass=group))";
     %ldapextrgroups

/*     filter="(&(name>=A)(name<=H)(objectClass=group))";       */
/*     %ldapextrgroups                                          */

     filter="(&(name>=H)(name<=P)(objectClass=group))";
     %ldapextrgroups

     filter="(&(name>=P)(objectClass=group))";
     %ldapextrgroups
```

```
     /* close connection to LDAP server */
     call ldaps_close(handle,rc);
     if rc NE 0 then do;
        msg = sysmsg();
        put msg;
     end;
run;


/*************************************************************************/
/* Sort the list of groups extracted from Active Directory by the     */
/* distinguishedName attribute which represents the actual Group      */
/* name. This is necessary so the following  datastep can do BY       */
/* processing on the Group list in order to detect the next unique    */
/* Group name and output it to &idgrptbl.                             */
/*************************************************************************/

proc sort data=&extractlibref..ldapgrps;
     by distinguishedName;
run;

proc datasets library=&extractlibref memtype=data;    /* Create Index */
     modify ldapgrps;                          /* for speedy retrieval */
     index create distinguishedName;
run;


/*****************************************************************************/
/* The following datastep creates the normalized tables for groups and group  */
/* membership from the &extractlibref..ldapusers extracted above.             */
/*****************************************************************************/

data &idgrptbla            /* Macros to define canonical Tables from %mduimpc */
     &idgrpmemstbla
      ;
     %defineidgrpcols;        /* Macros to define Table Columns from %mduimpc */
     %defineidgrpmemscols;

     set &extractlibref..ldapgrps;
         by distinguishedName;

     /*************************************************************************/
     /* When distinguishedName value changes set its column values and output */
     /* the next unique Group name to the Table of Groups: &idgrptbl.         */
     /*************************************************************************/
     if first.distinguishedName then do;
        keyid = distinguishedName;
        /* name already assigned from original       */
        /* description already assigned from original */
        grptype="" ;
        output &idgrptbl;
     end;

     /*************************************************************************/
     /* Each row in &extractlibref..ldapgrps represents membership in a Group so */
     /* set its column values and output unconditionally to &idgrpmemstbl.       */
     /*************************************************************************/

     grpkeyid=distinguishedName;
     memkeyid=member;
```

```
     output &idgrpmemstbl;
run;

/*****************************************************************************/
/* If we were using the employeeid as the keyid for persons, then we need to     */
/* re-code the person group memberkeys from DN to employeeIDs so that they match.*/
/*****************************************************************************/
%macro transmemkeyid;
   %if %upcase(&keyidvar)=EMPLOYEEID %then %do;
      proc sql;
         update &idgrpmemstbl
            set memkeyid =
                case when (select unique &keyidvar from &extractlibref..ldapusers
                              where memkeyid = distinguishedName)
                                 is missing then memkeyid
                     else (select unique &keyidvar from &extractlibref..ldapusers
                              where memkeyid = distinguishedName)
                end;
      quit;
   %end;
%mend;

%transmemkeyid;


/*****************************************************************************/
/* The idgrps and grpmems (i.e. group definitions and group membership) */
/* Tables are already in sorted order.  Create Indexes for them.        */
/*****************************************************************************/

proc datasets library=&importlibref memtype=data;   /* Create Index for */
     modify idgrps;                                 /* speedy retrieval */
     index create keyid;
run;

proc datasets library=&importlibref memtype=data;   /* Create Index for */
     modify grpmems;                                /* speedy retrieval */
     index create grpkeyid;
run;


/*****************************************************************************/
/* We've imported group membership without knowing if the members were  */
/* actually imported as people or groups.  If they weren't then we'll   */
/* get messages during the load about unknown group members.  To avoid  */
/* those messages, let's go ahead and eliminate those "unknown members. */
/*****************************************************************************/

proc sql;
     delete from &idgrpmemstbl
        where memkeyid not in (select unique keyid from &persontbl)
                 and memkeyid not in (select unique keyid from &idgrptbl);
quit;



/*****************************************************************************
 *****************************************************************************
 **                                                                         **
 **  SECTION 5: %mduimpl reads the canonical datasets, generates            **
```

```
 **             XML representing metadata objects, and invokes PROC        **
 **             METADATA to load the metadata.                             **
 **                                                                        **
 ****************************************************************************
 ****************************************************************************/

/****************************************************************************/
/* Change path for filename keepxml to your location                         */
/****************************************************************************/

%macro Execute_Load;

/* if the _EXTRACTONLY macro is set, then return and don't do any load processing.
*/
%if %symexist(_EXTRACTONLY) %then %return;

%mduimpl(libref=&importlibref,
         temp=work,
         outrequest=keepxml,
         outresponse=,
         submit=1,
         extidtag=&ADExtIDTag);
%mend Execute_Load;

%Execute_Load;
```