Paper 225-28

MVS Point-and-click Access to IMS Data with SAS/ACCESS®

Claude Rhéaume, Desjardins Financial Security, Québec, Canada
Gilles Turgeon, TELUS Solutions d'affaires, Québec, Canada

ABSTRACT

The purpose of this paper is to illustrate some techniques to access any IMS database with SAS.

Topics covered are :

- The basic requirements;
- What data is required in order to populate our METADATA;
- How to proceed;
- What is left for you to do;
- An interactive application to facilitate your work.

INTRODUCTION

In this paper, we wish to present the complete procedure for using the SAS Access To IMS product. For starters, we will enumerate all of the necessary requirements of this product, and briefly describe its use. Lastly, we will present an interactive, point-and-click tool, which we have developed under SAS AF, which makes it easier for the various users to take advantage of the possibilities alluded to in the first half of this paper. This tool can build Access Descriptors, INPUT statements based on COBOL copybooks, and finally generate a functional DATA STEP which responds to user input through our programming screen. As a result, the user only needs to complete the generated SAS code and then verify the results.

GETTING STARTED

First, identify the database(s) to be processed. Then, obtain the definitions (DBD) for these databases. You must also have a PSB (Program Scheduled Block) for your DBA (Database Administrator), which will allow you to establish which segments you can access and what actions (insert, delete, read, append) the program will be allowed to perform.

Each type of access upon a segment is identified as a PCB (Program Control Block). The sequential number of this PCB inside the PSB is the one you must specify upon reading (Infile statement).

EXAMPLES PROVIDED BY SAS

DBD EXAMPLE

```
DBD   NAME=ACCTDBD,ACCESS=(HDAM,OSAM),
            RMNAME=(DFSHDC40,3,71)
      DATASET
DD1=ACCTDD,DEVICE=<devicetype>,BLOCK=2400
      SEGM  NAME=CUSTOMER,PARENT=0,BYTES=225
      FIELD
NAME=(SSNUMBER,SEQ,U),BYTES=11,START=1,TYPE=C
      FIELD NAME=CUSTNAME,BYTES=40,START=12,TYPE=C
      FIELD NAME=CUSTADD1,BYTES=30,START=52,TYPE=C
      FIELD NAME=CUSTADD2,BYTES=30,START=82,TYPE=C
```

```
      FIELD NAME=CUSTCITY,BYTES=28,START=112,TYPE=C
      FIELD NAME=CUSTSTAT,BYTES=2,START=140,TYPE=C
      FIELD
NAME=CUSTLAND,BYTES=20,START=142,TYPE=C
      FIELD NAME=CUSTZIP,BYTES=10,START=162,TYPE=C
      FIELD
NAME=CUSTHPHN,BYTES=12,START=172,TYPE=C
      FIELD
NAME=CUSTOPHN,BYTES=12,START=184,TYPE=C
      SEGM
NAME=CHCKACCT,BYTES=40,PARENT=CUSTOMER
      FIELD
NAME=(ACNUMBER,SEQ,U),BYTES=12,START=1,TYPE=X
      FIELD NAME=STMTAMT,BYTES=5,START=13,TYPE=P
      FIELD NAME=STMTDATE,BYTES=6,START=18,TYPE=X
      FIELD NAME=STMTBAL,BYTES=5,START=26,TYPE=P
      SEGM  NAME=CHCKDEBT,BYTES=80,
          PARENT=((CHCKACCT,DBLE)),RULES=(,LAST)
      FIELD NAME=DEBTAMT,BYTES=5,START=1,TYPE=P
      FIELD NAME=DEBTDATE,BYTES=6,START=6,TYPE=X
      FIELD NAME=DEBTBLNK,BYTES=2,START=12,TYPE=X
      FIELD NAME=DEBTTIME,BYTES=8,START=14,TYPE=C
      FIELD NAME=DEBTDESC,BYTES=59,START=22,TYPE=C
      SEGM  NAME=CHCKCRDT,BYTES=80,
          PARENT=((CHCKACCT,DBLE)),RULES=(,LAST)
      FIELD NAME=CRDTAMT,BYTES=5,START=1,TYPE=P
      FIELD NAME=CRDTDATE,BYTES=6,START=6,TYPE=X
      FIELD NAME=CRDTBLNK,BYTES=2,START=12,TYPE=X
      FIELD NAME=CRDTTIME,BYTES=8,START=14,TYPE=C
      FIELD NAME=CRDTDESC,BYTES=59,START=22,TYPE=C
      SEGM
NAME=SAVEACCT,BYTES=40,PARENT=CUSTOMER
      FIELD
NAME=(ACNUMBER,SEQ,U),BYTES=12,START=1,TYPE=X
FIELD NAME=STMTAMT,BYTES=5,START=13,TYPE=P
FIELD NAME=STMTDATE,BYTES=6,START=18,TYPE=X
FIELD NAME=STMTBAL,BYTES=5,START=26,TYPE=P
SEGM  NAME=SAVEDEBT,BYTES=80,
    PARENT=((SAVEACCT,DBLE)),RULES=(,LAST)
FIELD NAME=DEBTAMT,BYTES=5,START=1,TYPE=P
FIELD NAME=DEBTDATE,BYTES=6,START=6,TYPE=X
FIELD NAME=DEBTBLNK,BYTES=2,START=12,TYPE=X
FIELD NAME=DEBTTIME,BYTES=8,START=14,TYPE=C
FIELD NAME=DEBTDESC,BYTES=59,START=22,TYPE=C
SEGM  NAME=SAVECRDT,BYTES=80,
    PARENT=((SAVEACCT,DBLE)),RULES=(,LAST)
FIELD NAME=CRDTAMT,BYTES=5,START=1,TYPE=P
FIELD NAME=CRDTDATE,BYTES=6,START=6,TYPE=X
FIELD NAME=CRDTBLNK,BYTES=2,START=12,TYPE=X
FIELD NAME=CRDTTIME,BYTES=8,START=14,TYPE=C
FIELD NAME=CRDTDESC,BYTES=59,START=22,TYPE=C
DBDGEN
```

PCB EXAMPLE

```
PCB
TYPE=DB,DBDNAME=ACCTDBD,PROCOPT=L,KEYLEN=23
```

```
SENSEG NAME=CUSTOMER,PARENT=0
SENSEG NAME=CHCKACCT,PARENT=CUSTOMER
SENSEG NAME=CHCKDEBT,PARENT=CHCKACCT
SENSEG NAME=CHCKCRDT,PARENT=CHCKACCT
SENSEG NAME=SAVEACCT,PARENT=CUSTOMER
SENSEG NAME=SAVEDEBT,PARENT=SAVEACCT
SENSEG NAME=SAVECRDT,PARENT=SAVEACCT
PSBGEN LANG=ASSEM,PSBNAME=ACCULOD
END
```

WHAT TO DO THE FIRST TIME BEFORE YOU ACCESS YOUR DATA WITH SAS ACCESS TO IMS

CREATION OF THE ACCESS DESCRIPTOR

This part must be done only once for each required database. It is critical that this work is done perfectly; success and failure of subsequent developments on this database using SAS Access To IMS will depend on the quality of your Access Descriptor. Thus, each field/segment must be correctly defined.

Certain tools are provided by SAS to help with this part of the work. They will allow us to automatically convert a DBD as well as a COBOL Copybook. These tools provide a good base but the user will have to verify and possibly complete this conversion manually.

First you must convert your DBD into an Access Descriptor. Please note that often, all fieds are not present in the DBD; the missing ones will have to be added to the Descriptor from the Copybook.

All necessary fields must be defined in this Access Descriptor so that they can be selected later.

Following this operation, you will obtain an access to your IMS database which can be understood by SAS. You can now take full advantage of your data.

THE ACCESS DESCRIPTOR AS USED IN OUR EXAMPLE

```
proc access dbms=ims;
  create mylib.account.access;
    dbd=acctdbd dbtype=hdam;
    record='customer_record' sg=customer sl=225;
      item=soc_sec_number        lv=2 dbf=$11.
key=u

se=ssnumber;
      item=customer_name         lv=2 dbf=$40.
se=custname;
      item='address info'        lv=2;
      item=addr_line_1           lv=3 dbf=$30.
se=custadd1;
      item=addr_line_2           lv=3 dbf=$30.
se=custadd2;
```

```
     item=city                  lv=3 dbf=$28.
se=custcity;
      item=state                 lv=3 dbf=$2.
se=custstat;
      item=country               lv=3 dbf=$20.
se=custland;
      item=zip_code              lv=3 dbf=$10.
se=custzip;
      item=home_phone            lv=2 dbf=$12.
se=custhphn;
      item=office_phone          lv=2 dbf=$12.
se=custophn;
record='checking_account_record' sg=chckacct
sl=40;
  item=check_account_number  lv=2 dbf=12.
key=u

se=acnumber;
  item=check_amount          lv=2 dbf=pd5.2
se=stmtamt  dbc=1;
  item=check_date            lv=2 dbf=6.0
fmt=date7.

se=stmtdate

dbc=mmddyy6.;
  item=filler1               lv=2 dbf=$2.;
  item=check_balance         lv=2 dbf=pd5.2
se=stmtbal  dbc=1;
record='checking_debit_record' sg=chckdebt
sl=80;
  item=check_debit_amount    lv=2 dbf=pd5.2
key=y

se=debtamt  dbc=1;
  item=check_debit_date      lv=2 dbf=6.0
se=debtdat  fmt=date7.

dbc=mmddyy6.;
  item=filler2               lv=2 dbf=$2.;
  item=check_debit_time      lv=2 dbf=$8.
se=debttime;
  item=check_debit_desc      lv=2 dbf=$59.
se=debtdesc;
record='checking_credit_record' sg=chckcrdt
sl=80;
  item=check_credit_amount   lv=2 dbf=pd5.2
key=y

se=crdtamt  dbc=1;
  item=check_credit_date     lv=2 dbf=6.0
se=crdtdate fmt=date7.

dbc=mmddyy6.;
  item=filler3               lv=2 dbf=$2.;
  item=check_credit_time     lv=2 dbf=$8.
se=crdttime;
  item=check_credit_desc     lv=2 dbf=$59.
se=crdtdesc;
record='savings_account_record' sg=saveacct
sl=40;
  item=savings_account_number lv=2 dbf=12.
key=u

se=acnumber;
  item=savings_amount        lv=2 dbf=pd5.2
se=stmtamt  dbc=1;
  item=savings_date          lv=2 dbf=6.0
se=stmtdate fmt=date7.

dbc=mmddyy6.;
  item=filler4               lv=2 dbf=$2.;
  item=savings_balance       lv=2 dbf=pd5.2
se=stmtbal  dbc=1;
```

```
     record='savings_debit_record' sg=savedebt
sl=80;
       item=savings_debit_amount  lv=2 dbf=pd5.2
key=y

se=debtamt  dbc=1;
       item=savings_debit_date    lv=2 dbf=6.0
```

```
se=debtdate fmt=date7.

dbc=mmddyy6.;
       item=filler5                lv=2 dbf=$2.;
       item=savings_debit_time     lv=2 dbf=$8.
se=debttime;
       item=savings_debit_desc     lv=2 dbf=$59.
se=debtdesc;

     record='savings_credit_record' sg=savecrdt
sl=80;
       item=savings_credit_amount lv=2 dbf=pd5.2
key=y

se=crdtamt  dbc=1;
       item=savings_credit_date   lv=2 dbf=6.0
se=crdtdate fmt=date7.

dbc=mmddyy6.;
       item=filler6                lv=2 dbf=$2.;
       item=savings_credit_time    lv=2 dbf=$8.
se=crdttime;
       item=savings_credit_desc    lv=2 dbf=$59.
se=crdtdesc;
   list all;
 run;
```

PRIMARY ELEMENTS OF AN ACCESS DESCRIPTOR

- *Record=* indicates the segment name;
- *Item=* indicates the field name;
- *Lv=* indicates the (COBOL-style) level for each item;
- *DBF=* indicates the Informat of each item;
- *SE=* is the search item, indicative of the items which are set as SEARCH FIELD inside the DBD;
- *Key=* indicates the index key as defined in the DBD.

DIFFERENT WAYS OF WORKING

WHAT CAN PROC ACCESS VIEW DO FOR US?

Proc Access View allows us to easily create a view over any database. We only have to identify the list of selected fields and add the appropriate WHERE statement. In little time, we get a SAS file that is ready to be used.

ACCESS VIEW EXAMPLE

```
proc sql;
  create view sql.charges as
    select distinct custacct.soc_sec_number,
                custacct.customer_name,
                custacct.check_account_number,
                chgdata.charge,
                bankchrg.chckchrg,
                bankchrg.atmfee,
                bankchrg.loanchrg
    from vlib.custacct,
        mydata.bankchrg,
        mydata.chgdata
    where custacct.soc_sec_number = bankchrg.ssn and
         custacct.check_account_number = chgdata.account;

  title2 'Banking Charges for the Month';
  select * from sql.charges;

Run ;
```

GETTING TO THE DATA STEP

Proc Access View can become fairly complex when there are several segments to be extracted. In this case, the use of DATA STEP is the preffered method. Access to the databases is made through the INFILE statement, to which we must specify the PSB and PCB. Later, and before each INPUT, we may specify the type of access to be performed, as well as the key (SSA). Once the return code is validated, the selected fields may be read.

DATA STEP EXAMPLE

```
Data work.custlist ;

    infile acctsam dli status=st pcbno=2;

    input @01 soc_sec_number $char11. ;

    if st ne ' ' Then
       abort ;
run ;
```

THE POINT-AND-CLICK SAS/AF APPLICATION

MAKING IT INTERACTIVE

In order to provide any user with simple access to any IMS database, we have created an interactive tool which automates many of the required steps.

1- All facilities mentioned here rely upon the creation of a metadatabase, that is, a SAS database containing all pertinent informations on every element, whether databases, segments, files, as well as each of the fields contained therein.

2- This tool makes sense out of a DBD and the COBOL Copybooks related to each segment, and then converts this data into SAS programming code. In the case of IMS segments, an Access Descriptor is generated, as well as the INPUT statement associated with every available segment from a given database. For VSAM and sequential databases, an INPUT statement is generated according to the user-selected fields.

3- Once the previous step has been performed by an admin who knows the data structures well, this tool will now allow any user to create their own program through an interactive interface. To that end, you must specify an application name, select the DBD or file to be used, and select the fields which you want to see appear in your statement. For IMS segments, you must also provide the required reading type, as well as the required PSB, PCB and SSA, if necessary.

4- Once these steps are completed, this tool will let you test in real time whether your code is correct. It will also let you generate a PSB which you can send to your DBA. Of course, it will also save the generated code in your selected directory, so that you can go back to it and complete it later if needed.

THE USER INTERFACE

So here are some snapshots of the tool we have created. Note that this application works in an OS/390 environment, i.e. in text mode. In order to improve its interface, we strongly recommend to configure your emulator so that the SELECT and ENTER functions be triggered by the right and left mouse buttons, respectively.

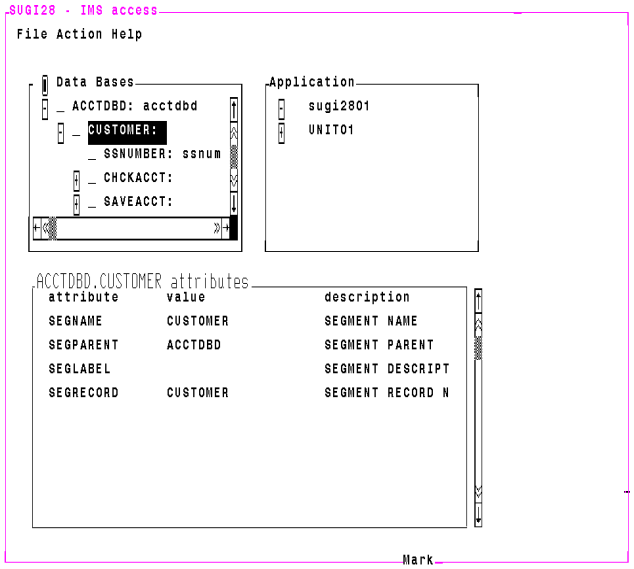
In the left frame, you may see all of the available databases under each heading, you can see an select any segment. Each selected elements will be move to the next frame.

In the middle frame, the user may visualize all unit, segment and item selected. Each unit means a INFILE statement at generation code time.

Finally, the right frame lists all attributes for any selected item in the database or user selected frame. At this place you can modify any authorized field to generate your own application.

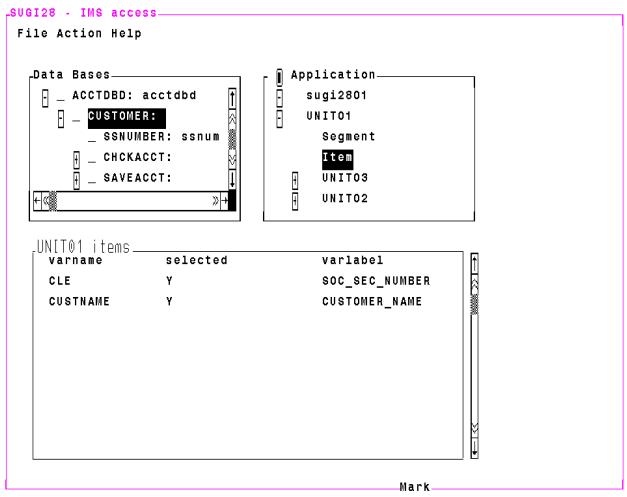
INTERACTIVE TOOL EXAMPLE 1

SNAPSHOT 1



INTERACTIVE TOOL EXAMPLE 2

SNAPSHOT 2



The PMENU at the top of the screen is to save, create or any standard action into your application. ACTION means to RUN or to VIEW your generate code and finally HELP can give you any helpful information.

EXAMPLE OF GENERATED DATA STEP

```
data TEST ;
  length ssa1 $9;

  infile acctsam dli pcbno=4 call=func
         ssa=ssa1 status=st;
  func = 'GN  ';
  ssa1 = 'CUSTOMER ';
  input @;
  if st ¬= '  ' and
     st ¬= 'CC' and
     st ¬= 'GA' and
     st ¬= 'GK' then
    abort;

  input @1 soc_sec_number $char11.;
  st = '  ';
  func = 'GNP ';
  ssa1 = 'CHCKACCT ';
do while (st = '  ');
  input @;
  if st = '  ' then
    do;
      input @13 check_amount pd5.2;
    end;
end;
if st ¬= 'GE' then
  abort;
st = '  ';
_error_ = 0;
```

```
input;
ssa1 = 'SAVEACCT ';
  do while (st = '  ');
    input @;
    if st = '  ' then
      do;
        input @13 savings_amount pd5.2;
      end;
  end;
  if st = 'GE' then
    _error_ = 0;
  else
    abort;



RUN;
```

INTENDED RESULT (PROC PRINT)

Customer Balances

OBS	chck_bal	save_bal	soc_sec-number
1	3005.60	784.29	667-73-8275
2	826.05	8406.00	434-62-1234
3	220.11	809.45	436-42-6394
4	2392.93	9552.43	434-62-1224
5	0.00	0.00	232-62-2432
6	1404.90	950.96	178-42-6534
7	0.00	0.00	131-73-2785
8	353.65	136.40	156-45-5672
9	1243.25	845.35	657-34-3245
10	7462.51	945.25	667-82-8275
11	608.24	929.24	456-45-3462
12	672.32	0.00	234-74-4612

CONCLUSION

SAS/Access to DLI is a particularly useful product for the processing of IMS databases. Views are easy to code, but they show their limits rapidly. Data steps offer a wider flexibility.

Our point-and-click application can generate the most efficient Data step code possible, and allows a user with limited IMS knowledge to accomplish these operations efficiently.

Since IMS access (INFILE and INPUT statements) is taken care of by this application, the programmer can concentrate on the other aspects of the project.

REFERENCES

SAS Institute, Inc.,
SAS/ACCESS Interface to IMS/DLI, Version 8,
Cary, NC: SAS Institute Inc., 2000

ACKNOWLEDGMENTS

We would like to thank David Turgeon for translation of this paper. The original text was written in French.

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® Indicates USA registration.

CONTACT INFORMATION

Please direct any questions or feedback to either of the authors at:

Claude Rhéaume
Desjardins Financial Security
Québec
Canada
Email: claude.rheaume@djsfc.com

Gilles Turgeon
TELUS Solutions d'affaires
Québec
Canada
Email: gilles.turgeon@telus.com