Paper 225-31

CEDA: The Invisible Hand

Bruce Gilsen, Federal Reserve Board

INTRODUCTION

Cross-Environment Data Access (CEDA) allows Version 7 or later SAS ® data sets created on any directory-based host supported by SAS such as Windows, Linux, or Solaris to be transparently read by the SAS System on another directory-based host. It can also be used after a directory-based host is upgraded from 32-bit to 64-bit to read the existing 32-bit data sets. Data sets from another directory-based host can be accessed from a Network File Services (NFS) mounted drive or copied to the new host with File Transfer Protocol (FTP) in binary mode. CEDA was introduced in Version 8, and has been part of Base SAS since Version 8.2.

CEDA provides many benefits, but it also important to understand its limitations. For example, it does not support pre-Version 7 data sets or most SAS objects other than data sets (catalogs, indexes, etc.), and does not allow a data set to be updated.

Because CEDA often operates transparently and because users do not precisely understand what constitutes updating a data set, CEDA can cause confusion. This paper tries to make the benefits and limitations of CEDA easy to understand.

NATIVE AND FOREIGN FORMAT: DEFINITION

A SAS library member has a *native format* if its data representation matches the host on which it is being processed. For example, a Windows SAS 32-bit data set has a native format on a Windows 32-bit host.

A SAS library member has a *foreign format* if its data representation does not match the host on which it is being processed. For example, a Windows SAS 32-bit data set has a foreign format on a Linux host or a Windows 64-bit host.

BENEFITS OF CEDA

Benefits include the following.

- CEDA operates transparently, so users can read a SAS data set without worrying about whether it has a native or foreign format.
- Reading a foreign data set with CEDA is faster than using a transport file. Reading with CEDA involves one translation, but using a transport file requires two translations - from SAS file to transport file on the original host, and from transport file to native SAS file on the destination host. In addition, users seem to make frequent mistakes working with transport files (at least at the Federal Reserve Board).
- Starting in Version 8.2, CEDA is part of Base SAS software. CEDA is available at all sites running Version 8.2 or later.

WHAT CEDA DOES AND DOES NOT SUPPORT

CEDA supports Version 7 or later SAS files of the following types: data sets, PROC SQL views, SAS/ACCESS views for Oracle and SYBASE, and multidimensional databases (MDDBs). The files must be created on a directory-based host. This paper focuses on data sets, since that will be of interest to most users.

CEDA does not support the following.

- Version 6 or earlier files. This includes Version 6 or earlier files created in Versions 7 and beyond (for example, with the V6 engine).
- Data set updates. See the sections *Data set updates: not supported by CEDA* and *Data set updates and CEDA: some reasons for the confusion* for details.
- Traditional bound libraries on z/OS mainframes. See the section *CEDA and the z/OS Mainframe* for details.
- Catalogs, DATA step views, or stored programs.

- Indexes, integrity constraints, or audit trails on data sets.

DATA SET UPDATES: NOT SUPPORTED BY CEDA

Updating a data set means changing a currently existing data set. This concept causes some confusion. For example, some users consider a DATA step or PROC COPY to be updating a data set, but they are actually replacing the data set or creating a additional data set, not updating the data set (an exception for the DATA step MODIFY statement is shown below).

CEDA does not support data set updates. As shown in this section, an error occurs when you try to update a data set that has a foreign format. The examples use Solaris and Linux-specific syntax, but apply to all directory-based hosts.

1. PROC FSEDIT, when the data set has a foreign format.

First, on Solaris, create a data set with one observation.

```
libname solaris1 '/my/home/m1bfg00/sasunix1';
data solaris1.one ;
  a=1; b=2;
run;
```

Then, on Linux, try to edit the data set with PROC FSEDIT.

```
libname linux1 '/my/home/m1bfg00/sasunix1';
proc fsedit data=linux1.one;
run;
```

The data set is readable but cannot be updated, so PROC FSEDIT opens in browse mode, not edit mode, and displays the following message in the FSBROWSE window.

```
NOTE: Using FSBROWSE since LINUX1.ONE cannot be opened for update.
```

Note that no message is displayed in the SAS log. Freeing the libref (libname solaris1 clear;) on Solaris or ending the Solaris SAS session has no impact, because the data set is not locked to the Linux session; it is in a foreign format to the Linux session.

2. PROC APPEND, when the BASE= data set (the data set being appended to) has a foreign format.

On Solaris, create a data set with one observation.

```
libname solaris1 '/my/home/m1bfg00/sasunix2';
data solaris1.one;
  a=1; b=2;
run;
```

Then, on Linux, create a data set with one observation, and try to append it to the data set created on Solaris.

```
libname linux1 '/my/home/m1bfg00/sasunix2';
data linux1.two;
  a=3; b=4;
run;
proc append base=linux1.one data=linux1.two;
run;
```

The following message is written to the SAS log.

```
ERROR: Updating not allowed for file LINUX1.ONE because it was created
       for a different operating system.
```

PROC APPEND updates only the BASE= data set, so the data set being appended (specified by DATA= in PROC APPEND) can be a foreign data set. On Linux, changing PROC APPEND to append the foreign data set (LINUX1.ONE) to the native data set (LINUX1.TWO) does work.

```
proc append base=linux1.two data=linux1.one;
run;
```

3. The DATA step MODIFY statement, when the data set has a foreign format.

Data set LINUX1.ONE was created in the previous example with a Solaris format. On Linux, submit the following code to modify variable A.

```
data linux1.one;
  modify linux1.one;
  a=a+100;
run;
```

The following message is written to the SAS log.

```
ERROR: Updating not allowed for file LINUX1.ONE because it was created
       for a different operating system.
```

4. The PROC DATASETS MODIFY statement, when the data set has a foreign format.

On Solaris, create a data set with one observation.

```
libname solaris1 '/my/home/m1bfg00/sasunix3';
data solaris1.one;
  date1=today();
run;
```

Then, on Linux, try to modify the data set by adding a format for variable DATE1.

```
libname linux1 '/my/home/m1bfg00/sasunix3';
proc datasets library=linux1 nolist;
  modify linux1.one;
  format date1 yymmddn8.;
quit;
```

The following message is written to the SAS log.

```
ERROR: Updating not allowed for file LINUX1.ONE because it was created
       for a different operating system.
```

DATA SET UPDATES AND CEDA: SOME REASONS FOR THE CONFUSION

1. Version 7 or later SAS data sets created on all directory-based hosts have the same extension - sas7bdat.

2. SAS data sets created on different directory-based hosts could reside in the same directory.

3. Many users are unclear on the precise concept of updating a data set. For example, consider the following DATA step and PROC step. Data sets ONE and TWO have four and two observations, respectively, and both contain numeric variables A and B.

```
data one;
  set one two;
run;

proc append
  base=one append=two;
run;
```

If successful, both steps create data set ONE, with six observations: four observations come from data set ONE followed by the two observations from data set TWO. Many users consider these steps the same, but they differ as follows.

DATA step.

The DATA statement causes a temporary data set to be created. SAS tries to write data set ONE's four observations followed by data set TWO's two observations to the temporary data set. If successful, the temporary data set replaces data set ONE. If unsuccessful, data set ONE is unchanged. This is not an update of data set ONE, because the observations are written to a temporary data set. If data sets ONE or TWO are foreign but readable with CEDA, the step executes successfully.

PROC APPEND.

SAS tries to add the two observations in data set TWO to the end of data set ONE. This is considered an update of data set ONE because the data set is changed but not replaced. If data set ONE has a foreign format, an error occurs. Since data set TWO is only read, it can be in either native or foreign format since it is readable through CEDA.

4. A foreign format data set could exist (and be read but not updated) for some time before an update was attempted, making the problem hard to identify.

CEDA AND THE Z/OS MAINFRAME

On the z/OS mainframe (and on OS/390, the predecessor of z/OS), CEDA does not work with SAS data sets in a traditional direct access or sequential access bound library, but does work with SAS data sets in a directory-based file system called the UNIX System Services (USS) Directory, also known as an HFS (hierarchical file system) library or HFS directory.

The HFS (or an equivalent Unix file system, zFS), is generally available on z/OS mainframes. At some sites, intervention by a systems administrator might be necessary prior to using HFS directories.

SAS System interaction with HFS directories is similar to directory interaction on other directory-based hosts, as follows.

- SAS data sets are stored as individual files in a directory with the standard extension, sas7bdat.
- SAS data sets can be copied with FTP or accessed from a Network File Services (NFS) mounted drive.
- Data set names can be longer than 8 characters and include upper and lower case characters.

The following SAS sessions provide a limited sense of how HFS directories are used. The specific steps or statements could differ at some sites. For more information, see the *SAS Companion for z/OS, Version 9.1.3* or z/OS documentation.

From Windows, remotely log on to the mainframe with FTP, then create an HFS directory if it does not already exist.

```
mkdir /u/m1bfg00/sastest1
```

Now, log on to the mainframe directly and submit a batch job to create data set ONE in the HFS directory.

In your JCL (use the proper case; /u/m1bfg00/sastest1 and /U/M1BFG00/SASTEST1 are different directories):

```
//BLAH     DD PATH='/u/m1bfg00/sastest1'
```

In your SAS code:

```
data blah.one;
  a=1;b=2;output;
  a=3;b=4;output;
run;
```

Back in Windows, display the name of the SAS data sets in the directory with DIR (in this case, just one.sas7bdat), then copy the SAS data set (in binary mode) from z/OS to Windows.

```
dir /u/m1bfg00/sastest1
cd /u/m1bfg00/sastest1
type binary
get one.sas7bdat
```

Now, in SAS for Windows, execute PROC CONTENTS for data set ONE, and the output includes the following.

```
Data Representation  MVS_32
```

One additional use of HFS directories with the SAS System, while peripheral to this paper, should be noted. Files (HTML files, PDF files, etc.) created on z/OS with the Output Delivery System (ODS) can be written to an HFS directory instead of an extended partitioned data set (PDSE). The advantages of writing these files to an HFS directory include the following.

- You can easily copy the entire directory to the destination host with FTP or another method.

- You can avoid renaming issues. For example, mywebpage1.htm is not a valid PDSE member name, so you must write it to a PDSE member with an abbreviated name and rename it on the destination host. Renaming issues can become increasingly difficult as the number of files increases.

CREATING A SAS DATA SET IN A FOREIGN FORMAT

To create a SAS data set in a foreign format, use the option OUTREP=*data-representation* either

- in a LIBNAME statement, when it applies to all data sets created using the libref from that LIBNAME statement
- as a data set option when creating a single foreign format data set

For a summary table of data representation values, see the table *Data Representation Values for OUTREP= Option* in the section *LIBNAME Statement* in the chapter *Statements* in the manual *SAS 9.1.3 Language Reference: Dictionary*. Some examples follow; they apply to all directory-based hosts.

1. On Windows, create Linux SAS data sets in the directory c:\sasdir1.

```
libname blah 'c:\sasdir1' outrep=linux_32;
```

2. On Windows, sort data set ONE, creating output data set TWO as a Linux SAS data set.

```
proc sort data=one out=two (outrep=linux_32);
  by x;
run;
```

3. On Linux, create the Windows SAS data set TWO in the library referenced by the libref BLAH.

```
data blah.two (outrep=windows);
```

4. On Windows, copy native (Windows) data sets NATIVE1.ONE and NATIVE1.TWO to foreign (Linux) data sets FOREIGN1.ONE and FOREIGN1.TWO. Other member types (such as catalogs) are not copied to the destination library. The NOCLONE option controls the values of the BUFFSIZE, COMPRESS, and REUSE data set attributes; see the *Base SAS 9.1.3 Procedures Guide* for details.

```
libname native1 ‘c:\sasdirn’;                    /* has native data sets ONE and TWO */
libname foreign1 ‘c:\sasdirf’ outrep=linux_32;   /* exists, is empty */
proc copy in=native1 out=foreign1 noclone;
run;
```

DETERMINING IF A DATA SET HAS A NATIVE OR FOREIGN FORMAT

This section shows several ways to determine if a data set has a native or foreign format. The examples are executed on Windows, using the following two data sets: WINDOWS1 with the native format, and LINUX1 with a foreign format (Linux in this case).

```
libname xxx ’c:\’;
data xxx.windows1;
  x=1;
run;
data xxx.linux1 (outrep=linux_32);
  y=1;
run;
```

1. Execute PROC CONTENTS.

Submit the following statements.

```
proc contents data=xxx.windows1; run;
proc contents data=xxx.linux1; run;
```

PROC CONTENTS output includes the following information.

For data set WINDOWS1: Data Representation WINDOWS_32
For data set LINUX1: Data Representation LINUX_32, INTEL_ABI

2. Execute PROC CONTENTS with the Output Delivery System (ODS).

The following code generates information for data set WINDOWS1.

```
ods output attributes=atr1(keep=member cvalue1 label1
    where=(label1 in ('Data Representation'))
    rename=(member=memname cvalue1=datarepname));
ods listing close;
proc contents data=xxx.windows1;
run;
ods listing;
proc print data=atr1;
  var memname datarepname;
run;
```

A one observation data set similar to the following is generated.

```
Obs   memname         datarepname

1     XXX.WINDOWS1    WINDOWS_32
```

For data set LINUX1, change the DATA= keyword in PROC CONTENTS statement from xxx.windows1 to xxx.linux1. In the resulting data set, the value of DATAREPNAME is LINUX_32, INTEL_ABI instead of WINDOWS_32.

3. Print data representation information from the data dictionary tables.

```
proc print data=sashelp.vtable;
  where libname = "XXX" and memname in ("WINDOWS1", "LINUX1");
  var memname datarep datarepname;
run;
```

The following output is generated.

```
Obs    memname    datarep    datarepname

  1    WINDOWS1   NATIVE     WINDOWS_32
  2    LINUX1     FOREIGN    LINUX_32, INTEL_ABI
```

4. Read data representation information from the data dictionary tables and create a SAS data set, then print the data set. The following code creates data set NEW with two observations. PROC PRINT generates the same output as method 3.

```
proc sql ;
  create table new as
  select memname, datarep, datarepname
  from dictionary.tables
  where libname = "XXX" and memname in ("WINDOWS1", "LINUX1");
quit;
proc print data=new;
run;
```

5. Read data representation information from the data dictionary tables and create macro variables that can be tested in a macro or elsewhere.

The following code creates macro variables DATAREP_ONE and DATAREPNAME_ONE containing the DATAREP and DATAREPNAME information for data set XXX.WINDOWS1, and displays the values.

```
proc sql noprint;
  select datarep, datarepname
  into :datarep_one,:datarepname_one
  from dictionary.tables
  where libname="XXX"
  and memname="WINDOWS1";
quit;
```

```
   %put datarep_one= &datarep_one;
   %put datarepname_one= &datarepname_one;
```

The following text is written to the SAS log.

```
   datarep_one= NATIVE
   datarepname_one= WINDOWS_32
```

Notes about determining if a data set has a native or foreign format.

- Version 9 PROC CONTENTS output differs from Version 8 PROC CONTENTS output.
- The data set created by PROC CONTENTS with the OUT= keyword does not contain data representation information. To obtain this information, create an ODS output data set containing the CONTENTS output, as shown in example 2 above.
- The data dictionary tables provide both DATAREPNAME (data representation value) and DATAREP (NATIVE/FOREIGN). PROC CONTENTS provides only DATAREPNAME.
- The CONTENTS statement in PROC DATASETS is equivalent to PROC CONTENTS.
- For a summary table of data representation values, see the table *Data Representation Values for OUTREP= Option* in the section *LIBNAME Statement* in the chapter *Statements* in the manual *SAS 9.1.3 Language Reference: Dictionary*.

MONITORING USE OF CEDA

1. In Version 9, prior to Version 9.1.3 Service Pack 3.

Set the SAS System Option MSGLEVEL to I instead of the default value, N, to request that SAS write a message to the log when CEDA is used. Additional messages about merges, sorts, and indexes are also generated.

```
   options msglevel=i;run;
```

After setting OPTIONS MSGLEVEL=i;, the following message was written to the log when foreign data set LINUX1 was read in Version 9.1.3 of SAS for Windows.

```
   INFO: Data file WORK.LINUX1.DATA is in a format native to another host or the file
   encoding does not match the session encoding. Cross Environment Data Access will be
   used, which may require additional CPU resources and reduce performance.
```

2. Starting with Version 9.1.3 Service Pack 3, released in mid-2005.

SAS writes a message to the log when CEDA is used if either of the following is true.

- The SAS System Option MSGLEVEL is set to I, as before.
- The SAS System Option NOTES is in effect.

CEDA AND PERFORMANCE ISSUES

Reading a foreign data set with CEDA requires a translation to the native format in memory. Reading a large foreign data set multiple times could degrade performance. To improve performance, use the OUTREP= option to create the data set in the native format of the host on which it will be read, or convert the data set to a native file after it is copied to the new host.

The performance of native and foreign SAS data sets was compared by reading native and foreign copies of a data set containing 5,000, 100,000, 1,000,000, and 5,000,000 observations with SAS Version 9.1.3 for Windows.

The following program was used to create the data sets. OUTREP=LINUX was included in the first DATA statement to create a foreign data set, and omitted otherwise. The upper bound of the DO loop was used to vary the number of observations.

```
   libname windows1 'c:\testceda';
   *data windows1.oldfile (outrep=linux);   /* either create a foreign file */
```

```
data windows1.oldfile ;                  /* or create a native file */
  do i = 1 to 5000;   /* upper loop bound: one of 5000, 100,000, 1,000,000, 5,000,000 */
    v1=i;
    v2=i+1;
    v3=i+2;
    v4=i+3;
    v5=i+4;
    v6=i+5;
    v7=i+6;
    v8=i+7;
    v9=i+8;
    v10=i+9;
    output;
  end;
run;
data newfile;
  set windows1.oldfile;
  v11=v10-v9+v8-v7+v6-v5+v4-v3+v2-v1;
run;
```

The following table shows the CPU time needed to create DATA set NEWFILE.

```
Observations    Native CPU seconds   Foreign CPU seconds

    5,000          .03                    .03
  100,000          .15                    .29
1,000,000         1.40                   2.78
5,000,000         6.63                  13.37
```

On Unix platforms, an additional consideration is that 32-bit and 64-bit data sets differ only in the header; the data portions of the data sets are the same. Consider the following cases.

- On a Linux host, reading a Windows data set requires CEDA to translate both the header and the data portion of the data set.
- On a Linux 64-bit host, reading a 32-bit Linux data set requires CEDA to translate only the header, and the performance loss from using CEDA is much less than noted above.

CONVERTING FOREIGN MEMBERS TO NATIVE MEMBERS

Several methods can be used to convert foreign library members to native members. The appropriate method depends upon the types of members that are present, including members (such as catalogs) not supported by CEDA. A complete review of all possible scenarios is beyond the scope of this paper; this section presents a few common solutions. For more details, see SAS documentation and SAS Institute's Migration Community web site at http://support.sas.com/rnd/migration/.

In this section, assume that data sets are supported by CEDA unless otherwise noted (i.e., they are on a directory-based host and are Version 8 or later data sets).

1. DATA step.

This method can be used to convert a single data set in Version 8 or 9. In this example, native data set NATIVE1.ONE is created from foreign data set FOREIGN1.ONE. Some attributes of the original data set, such as compression, passwords, or buffers, could be lost.

```
libname foreign1 ‘c:\sasdirf’;   /* has foreign data set ONE */
libname native1 ‘c:\sasdirn’;    /* exists, is empty */
data native1.one;
  set foreign1.one;
run;
```

2. PROC COPY.

This method can be used to convert one or more data sets in Version 8 or 9. In this example, native data sets NATIVE1.ONE and NATIVE1.TWO are created from foreign data sets FOREIGN1.ONE and FOREIGN1.TWO. Other member types (such as catalogs) are copied if they have the same data representation as the current host.

```
libname foreign1 ‘c:\sasdirf’;   /* has foreign data sets ONE and TWO */
libname native1 ‘c:\sasdirn’;    /* exists, is empty */
proc copy in=foreign1 out=native1;
run;
```

3. PROC CPORT and PROC CIMPORT.

This method can be used to convert data sets, catalogs, and indexes in Version 8 or 9. Use of the SAS System on both the original and destination hosts is required. First, PROC CPORT is executed on the original host, creating a transport file. Then, the transport file is transferred in binary mode to a drive that is accessible on the destination host. Finally, PROC CIMPORT is executed on the destination host, creating SAS data sets, catalogs, and indexes that are native to the destination host. An example of converting SAS library members from Solaris to Windows follows.

On the original host, Solaris, write the data sets, catalogs, and indexes in the SAS library /my/home/m1bfg00/sasdirf to the transport file /my/home/m1bfg00/transfile.

```
libname solaris1 ‘/my/home/m1bfg00/sasdirf’;   /* has Solaris data sets,
                                                  catalogs, indexes */
filename trans ‘/my/home/m1bfg00/transfile’;   /* does not exist */
proc cport library=solaris1 file=trans;
run;
```

Transfer the transport file in binary mode to the file c:\transfile on the destination host, Windows. On Windows, execute PROC CIMPORT to create SAS data sets, catalogs, and indexes native to Windows in the directory c:\sasdirn.

```
filename trans ‘c:\transfile’;   /* transport file */
libname windows1 ‘c:\sasdirn’;   /* exists, is empty */
proc cimport file=trans library=windows1;
run;
```

4. PROC MIGRATE.

PROC MIGRATE is available starting in Version 9.1. It is designed to migrate older SAS libraries to the current release. It supports SAS library members from the later releases of Version 6 (for example, Versions 6.09, 6.11, and 6.12 on Solaris) and beyond. PROC MIGRATE is not meant to be used cross-host; it is intended for use within the same operating system family.

PROC MIGRATE supports the following member types:

- SAS/ACCESS access views and descriptors (Version 6.12 or later)
- catalogs (some special handling might be required to migrate 32-bit catalogs to 64-bit catalogs; see Olson and Wiehle (2003) for details)
- data sets
- DATA step views (from Version 8 and later when the source was saved in the view)
- item stores (with a few exceptions)
- MDDB files (except for Version 7 MDDB files)
- SQL views

PROC MIGRATE provides great flexibility but care must be taken with regard to the exceptions noted above.

PROC MIGRATE is documented in the *Base SAS 9.1.3 Procedures Guide*. A great deal of additional information about PROC MIGRATE and migration issues is available on SAS Institute’s Migration Community web site at http://support.sas.com/rnd/migration/.

Here is a simple example of using PROC MIGRATE to migrate a library. The directory c:\sasdir6 contains Version 6.12 Windows SAS library members supported by PROC MIGRATE (see list above). PROC MIGRATE creates Version 9 Windows SAS library members in the directory c:\sasdir9.

```
libname vers6lib v6 ‘c:\sasdir6’;     /* has Version 6.12 library members */
libname vers9lib base ‘c:\sasdir9’;   /* exists, is empty */
proc migrate in=vers6lib out=vers9lib;
run;
```

Notes on converting foreign members to native members.

- With some modification, these methods work for all but the oldest Version 6 files with the same data representation as the current host. For example, on Solaris, these methods work for Solaris data sets from Versions 6.09, 6.11, or 6.12 (extension ssd01) but not from Version 6.03 (extension ssd), and only method 2 (PROC COPY) works for Solaris data sets from Version 6.07 (extension ssd01). Mixed directories (containing files from Versions 6 and 8) require one LIBNAME statement and one PROC for each version.

- The examples apply to all directory-based hosts.

- To convert native members to foreign members, use a DATA step or PROC COPY with the OUTREP= option. See the section *Creating a SAS data set in a Foreign Format* for details.

CONCLUSION

CEDA allows transparent processing of SAS library members that have a foreign format. This simplifies the use of the SAS System, but can cause confusion in some cases. By showing the uses and limitations of CEDA, this paper has hopefully made CEDA easier to understand.

For more information, contact the author, Bruce Gilsen, one of the following ways.

mail: Federal Reserve Board, Mail Stop 157, Washington, DC 20551
phone: 202-452-2494
e-mail: bruce.gilsen@frb.gov

REFERENCES

Olson, Diane and Wiehle, David (2003), "PROC MIGRATE: How to Migrate Your Data and Know You’ve Done It Right!,” *Proceedings of the Twenty-Eighth Annual SAS Users Group International Conference*.

SAS Institute Inc (2004), "*Base SAS 9.1.3 Procedures Guide*," Cary, NC: SAS Institute Inc.

SAS Institute Inc (2004), "*Moving and Accessing SAS 9.1 Files*," Cary, NC: SAS Institute Inc.

SAS Institute Inc (2004), "*SAS 9.1.3 Companion for z/OS*," Cary, NC: SAS Institute Inc.

SAS Institute Inc (2004), "*SAS 9.1.3 Language Reference: Concepts*," Cary, NC: SAS Institute Inc.

SAS Institute Inc (2004), "*SAS 9.1.3 Language Reference: Dictionary, Volumes 1, 2, and 3*," Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

The following people contributed extensively to the development of this paper: Donna Hill and Steve Taubman at the Federal Reserve Board, Diane Olson at SAS Institute, and Christianna Williams at the University of North Carolina at Chapel Hill. Their support is greatly appreciated.

TRADEMARK INFORMATION