CHAPTER

5

ACCESS Procedure Reference

Introduction

The ACCESS procedure enables you to create and edit descriptor files used by the SAS/ACCESS interface to ADABAS. This chapter provides reference information for the ACCESS procedure statements, including procedure syntax and statement options.

Additionally, the following sections provide information to help you optimize the use of the interface:

- "Creating and Using View Descriptors Efficiently" on page 98 presents efficiency considerations for using the SAS/ACCESS interface to ADABAS

- "ACCESS Procedure Formats and Informats" on page 98 summarizes how the SAS/ACCESS interface converts each type of ADABAS data into its equivalent SAS variable format.
- "Effects of the SAS/ACCESS Interface on ADABAS Data" on page 101 explains how the SAS/ACCESS interface handles specific ADABAS data fields.

If you need help with SAS data sets and data libraries, their naming conventions, or any terms used in regard to the ACCESS procedure, refer to the *SAS Language Reference: Dictionary* and the *SAS Companion for the MVS Environment, Version 6, Second Edition* .

Case Sensitivity in the ACCESS Procedure

SAS names are not case sensitive; they can be entered in either uppercase or lowercase. The ACCESS procedure converts DBMS column names to uppercase including names enclosed in quotes. Any DBMS names that contain special or national characters must be enclosed in quotes.

ACCESS Procedure Syntax

PROC ACCESS <*options*>;

Creating and Updating Statements

CREATE *libref.member-name*.ACCESS | VIEW;

UPDATE *libref.member-name*.ACCESS|VIEW <*password-level=SAS-password*>;

Database-Description Statements

DDM = *data-definition-module-name*;

NSS (LIBRARY | LIB= *library-identifier*
USER= *user-identifier*
PASSWORD | PW= *Natural-Security-password*);

ADBFILE (NUMBER | NUM= *Adabas-file-number*
PASSWORD | PW= *Adabas-password*
CIPHER | CC= *Adabas-cipher-code*
DBID= *Adabas-database-identifier*);

SYSFILE (NUMBER | NUM= *Adabas-system-file-number*
PASSWORD | PW= *Adabas-password*
CIPHER | CC= *Adabas-cipher-code*
DBID= *Adabas-database-identifier*);

SECFILE (NUMBER | NUM= *Natural-Security-system-file-number*
PASSWORD | PW= *Adabas-password*
CIPHER | CC= *Adabas-cipher-code*
DBID= *Adabas-database-identifier*);

Editing Statements

ASSIGN <=> YES | NO | Y | N;

CONTENT *column-identifier-1 <=> SAS-date-format | length | E*
<... column-identifier-n <=> SAS-date-format | length | E >;
DROP *column-identifier-1 <... column-identifier-n>*;
EXTEND <ALL | VIEW | *column-identifier-1 <... column-identifier-n>>*;
FORMAT *column-identifier-1 <=> SAS-format-name*
<...column-identifier-n <=> SAS-format-name>;
INFORMAT *column-identifier-1 <=> SAS-format-name*
<... column-identifier-n <=> SAS-format-name>;
KEY<=> *column-identifier-1 <...column-identifier-n>*;
LIST <ALL | VIEW | *column-identifier-1 <...column-identifier-n>>*;
LISTINFO <ALL | VIEW | *column-identifier-1 <...column-identifier-n>>*;
LISTOCC *column-identifier-1 <... column-identifier-n>*;
MVF *column-identifier*
CONTENT *occurrence-1 <=> SAS-date-format | length* | E
<... occurrence-n <=> SAS-date-format| length| E>;
|
DROP *occurrence-1* <<TO>*... occurrence-n>*;
|
FORMAT *occurrence-1 <=> SAS-format-name*
<... occurrence-n <=> SAS-format-name>;
|
INFORMAT *occurrence-1 <=> SAS-format-name*
<... occurrence-n <=> SAS-format-name>;
|
OCCURS <=> *number-of-occurrences*;
|
RENAME *occurrence-1 <=> SAS-variable-name*
<...occurrence-n <=> SAS-variable-name>;
|
RESET *occurrence-1* <<TO>*... occurrence-n>*;
|
SELECT *occurrence-1* <<TO>*... occurrence-n>*;
RENAME *column-identifier-1 <=> SAS-variable-name*
<... column-identifier-n <=> SAS-variable-name>;
RESET ALL | *column-identifier-1 <... column-identifier-n>*;
SECURITY <=> YES | NO | Y | N;
SELECT ALL | *column-identifier-1 <... column-identifier-n>*;
SUBSET *selection-criteria*;
QUIT;

RUN;

Description

You use the ACCESS procedure to create and edit access descriptors and view descriptors, and to create SAS data files. Descriptor files describe DBMS data so that you can read, update, or extract the DBMS data directly from within a SAS session or in a SAS program.

The ACCESS procedure runs in interactive line and batch modes. The following sections provide complete information on PROC ACCESS options and statements.

PROC ACCESS Statement Options

The ACCESS procedure statement takes the following options:

PROC ACCESS *options*;

Depending on which options you use, the ACCESS procedure statement performs several tasks.

You use the PROC ACCESS statement with database-description statements and certain procedure statements to create descriptors or SAS data files from DBMS data. See "Procedure Statements" on page 64 for information on which procedure statements to use for each task. The following sections describe PROC ACCESS options in greater detail.

Options

This section describes the options that you use to create and edit access descriptors and view descriptors.

ACCDESC=*libref.access-descriptor*
specifies an access descriptor.

ACCDESC= is used with the DBMS= option to create a view descriptor that is based on the specified access descriptor. You specify the view descriptor's name in the CREATE statement. You can also use a SAS data set option on the ACCDESC= option to specify any passwords that have been assigned to the access descriptor.

The ACCDESC= option has two aliases: AD= and ACCESS=.

DBMS=*ADABAS*
specifies which database management system you want to use. DBMS= can be used with the ACCDESC= option to create a view descriptor, which is then named in the CREATE statement.

OUT=<*libref.*>*member-name*
specifies the SAS data file to which DBMS data are written. OUT= is used only with the VIEWDESC= option.

VIEWDESC=<*libref.*>*view-descriptor*
specifies a view-descriptor that accesses the ADABAS data. VIEWDESC= is used only with the OUT= option.

For example:

```
proc access dbms=adabas viewdesc=vlib.invq4
            out=dlib.invq4;
run;
```

The VIEWDESC= option has two aliases: VD= and VIEW=.

CAUTION:
Altering a DBMS table can invalidate descriptors. Altering the format of a DBMS table that has descriptor files defined on it might cause these descriptors to be out-of-date or no longer valid. For example, if you add a column to a table and an existing access descriptor is defined on that table, the access descriptor and any view descriptors based on it do not show the new column. You must re-create the descriptors to be able to show and select the new column. △

SAS System Passwords for SAS/ACCESS Descriptors

The SAS System enables you to control access to SAS data sets and access descriptors by associating one or more SAS System passwords with them. You must first create the descriptor files before assigning SAS passwords to them.

Table 5.1 on page 63 summarizes the levels of protection that SAS System passwords have and their effects on access descriptors and view descriptors:

Table 5.1 Password and Descriptor Interaction

	READ=	WRITE=	ALTER=
access descriptor	no effect on descriptor	no effect on descriptor	protects descriptor from being read or edited
view descriptor	protects DBMS data from being read or updated	protects DBMS data from being updated	protects descriptor from being read or edited

When you create view descriptors, you can use a SAS data set option after the ACCDESC= option to specify the access descriptor's password (if one exists). In this case, you are *not* assigning a password to the view descriptor that is being created; rather, using the password grants you permission to use the access descriptor to create the view descriptor. For example:

```
proc access dbms=ababas accdesc=adlib.customer
            (alter=rouge);
  create vlib.customer.view;
  select all;
run;
```

By specifying the ALTER-level password, you can read the ADLIB.CUSTOMER access descriptor and therefore create the VLIB.CUSTOMER view descriptor.

For detailed information on the levels of protection and the types of passwords you can use, refer to the *SAS Language Reference: Dictionary*. The following section describes how you assign SAS System passwords to descriptors.

Assigning Passwords

To assign, change, or clear a password for an access descriptor, a view descriptor, or another SAS file, use the DATASETS procedure.

DATASETS Procedure Method

To assign, change, or delete a SAS password, use the DATASETS procedure's MODIFY statement in the PROGRAM EDITOR window. Here is the basic syntax for using PROC DATASETS to assign a password to an access descriptor, a view descriptor, or a SAS data file:

PROC DATASETS LIBRARY=*libref* MEMTYPE=*member-type*;
MODIFY *member-name* (*password-level* = *password-modification*);

RUN;

In this syntax statement, the *password-level* argument can have one or more of the following values: READ=, WRITE=, ALTER=, or PW=. PW= assigns read, write, and alter privileges to a descriptor or data file. The *password-modification* argument enables you to assign a new password or to change or delete an existing password.

For example, this PROC DATASETS statement assigns the password MONEY with the ALTER level of protection to the access descriptor ADLIB.SALARIES.

```
proc datasets library=adlib memtype=access;
   modify salaries (alter=money);
run;
```

In this case, users are prompted for the password whenever they try to browse or edit the access descriptor or to create view descriptors that are based on ADLIB.SALARIES.

You can assign multiple levels of protection to a descriptor or SAS data file. However, for more than one level of protection (that is, both READ and ALTER), be sure to use a different password for each level. If you use the same password for each level, a user to whom you grant READ privileges only (in order to read the DBMS data) would also have privileges to alter your descriptor (which you do not want to allow).

In the next example, the PROC DATASETS statement assigns the passwords MYPW and MYDEPT with READ and ALTER levels of protection to the view descriptor VLIB.JOBC204:

```
proc datasets library=vlib memtype=view;
   modify jobc204 (read=mypw alter=mydept);
run;
```

In this case, users are prompted for the SAS password when they try to read the DBMS data, or try to browse or edit the view descriptor VLIB.JOBC204 itself. You need both levels to protect the data and descriptor from being read. However, a user could still update the data accessed by VLIB.JOBC204, for example, by using a PROC SQL UPDATE. Assign a WRITE level of protection to prevent data updates.

To delete a password on an access descriptor or any SAS data set, put a slash after the password:

```
proc datasets library=vlib memtype=view;
   modify jobc204 (read=mypw/ alter=mydept/);
run;
```

Refer to the *SAS Language Reference: Dictionary* for more examples of assigning, changing, deleting, and using SAS System passwords.

Procedure Statements

To invoke the ACCESS procedure you use the options described in “Options” on page 62 and certain procedure statements. The options and statements that you choose are determined by your task.

□ To create an access descriptor:

PROC ACCESS DBMS=*ADABAS*;
CREATE *libref.member-name*.ACCESS;
required database-description statements;
optional editing statements;

RUN;

□ To create an access descriptor and a view descriptor:

```
PROC ACCESS DBMS=ADABAS;
  CREATE libref.member-name.ACCESS;
    required database-description statements;
    optional editing statements;
  CREATE libref.member-name.VIEW;
    SELECT item-list;
    optional editing statements;

RUN;
```

□ To create a view descriptor from an existing access descriptor:

```
PROC ACCESS DBMS=ADABAS ACCDESC=libref.access-descriptor;
  CREATE libref.member-name.VIEW;
    SELECT item-list;
    optional editing statements;

RUN;
```

□ To update an access descriptor:

```
PROC ACCESS DBMS=ADABAS;
  UPDATE libref.member-name.ACCESS;
    procedure statements;

RUN;
```

□ To update a view descriptor:

```
PROC ACCESS DBMS=ADABAS;
  UPDATE libref.member-name.VIEW;
    procedure statements;

RUN;
```

Dictionary

ADBFILE

Specifies the file number of the ADABAS file to be accessed.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: DDM, SECURITY

Syntax

ADBFILE (NUMBER | NUM = *Adabas-file-number*

PASSWORD | PW = *Adabas-password*
CIPHER|CC = *Adabas-cipher-code*
DBID = *Adabas-database-identifier*);

Details

The ADBFILE statement allows you to specify an ADABAS file number and optional password, cipher code, and database identifier for the ADABAS file to be used when reading the access descriptor. If you specified a NATURAL DDM using the DDM= statement in an access descriptor, then the file number is supplied by the DDM and the ADBFILE statement is not needed.

If you specified SECURITY=YES in the access descriptor, you cannot change the values for the password and cipher code in the view descriptor. However, if no values were entered in the access descriptor, you can enter them in the view descriptor, even if the SECURITY=YES statement has been issued.

Number
is the ADABAS file number of the file to be accessed. The ADABAS file number is a number from 1 to 255 that is assigned when the ADABAS files are created with the ADABAS ADACMP utility.

Password
is an ADABAS password, which provides security protection at the file or data-field level, or on the basis of a value at the logical-record level. The value is not displayed as you enter it, and it is written to the access descriptor in encrypted form.

Cipher Code
is an ADABAS cipher code, which is a numeric code for ciphering and deciphering data into and from an ADABAS file. The value is not displayed as you enter it, and it is written to the access descriptor in encrypted form.

DBID
is the ADABAS database identifier (number) to be accessed. The database identifier is a numerical value from 1 to 255 that is assigned to each ADABAS database.

ASSIGN

Indicates whether SAS variable names and formats are automatically generated.

Optional statement

Applies to: access descriptor

Interacts with: CONTENT, FORMAT, INFORMAT, KEY, MVF, RENAME, RESET

Default: NO

Syntax

ASSIGN<=>YES | NO | Y | N;

Details

The ASSIGN statement indicates whether SAS variable names are automatically generated and whether users can change SAS variable names and other column information the view descriptors created from this access descriptor.

An editing statement, such as ASSIGN, must be specified after the CREATE and database-description statements when you create an access descriptor. See "CREATE" on page 68 for more information.

The value **NO** (or **N**) enables you to modify SAS variable names, formats, informats, database contents, occurrence ranges, and BY keys when you create an access descriptor and when you create view descriptors that are based on this access descriptor.

Specify a **YES** (or **Y**) value for this statement to generate unique SAS variable names from the first eight characters of the DBMS column names, according to the rules listed below. With **YES**, you can change the SAS variable names and other column information only in the access descriptor. The SAS variable names and other column information that are saved in an access descriptor are *always* used when view descriptors are created from the access descriptor; you cannot change them in the view descriptors.

Default SAS variable names are generated according to these rules:

- If the column name is longer than eight characters, the SAS System uses only the first eight characters. If truncating results in duplicate names, numbers are appended to the ends of the names. For example, the DBMS names **clientsname** and **clientsnumber** become the SAS names **clientsn** and **clients1**.

  If the same descriptor has another set of columns with duplicate names, the numeric suffix begins at the next highest number from the previous set of duplicate names. For example, if the descriptor has the duplicate names above and also has the DBMS names **customername**, **customernumber**, and **customernode**, the default SAS names would be **customer**, **custome1**, and **custome2**.
- If the column name contains characters that are not valid in SAS names (including national characters), the SAS System replaces these characters with underscores (_). For example, the column name **func$** becomes the SAS variable name **func_**.

If you specify **YES** for this statement, the SAS System automatically resolves any duplicate variable names. However, if you specify **YES**, you cannot specify the CONTENT, FORMAT, INFORMAT, KEY, MVF (with OCCURS option), RENAME, or RESET statements when you create view descriptors that are based on the access descriptor.

When the SAS/ACCESS interface encounters the next CREATE statement to create an access descriptor, the ASSIGN statement is reset to the default **NO** value.

AN is the alias for the ASSIGN statement.

CONTENT

Specifies a SAS date format or length.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN

Syntax

CONTENT *column-identifier-1 <=> SAS-date-format | length | E*

<... column-identifier-n <=> SAS-date-format |length|E >;

Details

The CONTENT statement enables you to enter a SAS date format, a variable length, or an extended time format. A date format means that the ADABAS data have the specified representation. A variable length determines the number of characters to be accessed. The extended time format (E) invokes NATURAL date, time, and datetime values. The SAS System stores datetime values as the number of days and seconds before and after January 1, 1960. The NATURAL 4th generation language stores date and time values as the number of days and seconds since 0 A.D.

For ADABAS files, entering a SAS date or a variable length automatically changes default values for SAS formats and informats. For NATURAL DDMs, entering a date changes the default format and informat but entering a length does not. However, if you have previously changed any format and informat values, specifying a CONTENT value does not alter those values. Specifying extended time format changes default values for SAS informat and format values to DATETIME16.

For groups and periodic groups, the CONTENT field is for information only and is set to *GROUP* and *PGROUP*, respectively.

ADABAS does not have a specific date type; therefore, the CONTENT statement enables you to identify dates for SAS processing. You can enter one of four SAS date formats.

- YYMMDD*w*. where *w* is 6 for two-digit years or 8 for four-digit years
- MMDDYY*w*. where *w* is 6 for two-digit years or 8 for four-digit years
- DDMMYY*w*. where *w* is 6 for two-digit years or 8 for four-digit years
- JULIAN*w*. where *w* is 5 for two-digit years or 7 for four-digit years.

If you specified Assign=YES when creating an access descriptor, you cannot change the value for this statement when you later create a view descriptor based on that access descriptor. If you specified Assign=NO, you can change the value for this statement in a subsequent view descriptor.

You do not have to issue a SELECT statement for columns named in the CONTENT statement.

Note: The SAS/ACCESS to ADABAS engine does not provide automatic conversion to the extended time format in releases of the SAS System prior to Release 6.08 TSO420. However, it is possible to convert a value to the extended time format in a SAS DATA step by using the following formulas:

```
SAS date value     = NATURAL date value – 715874
SAS datetime value = (NATURAL datetime value / 10)
                        – (715874 * 3600 *24)
SAS time value     = NATURAL time value / 10
```

△

CREATE

Creates a SAS/ACCESS descriptor file.

Required statement

Applies to: access descriptor or view descriptor

Syntax

CREATE *libref.member-name*.ACCESS | VIEW;

Details The CREATE statement identifies the access descriptor or view descriptor that you want to create. This statement is required for creating a descriptor.

To create a descriptor, use a three-level name. The first level identifies the libref of the SAS data library where you will store the descriptor. You can store the descriptor in a temporary (WORK) or permanent SAS data library. The second level is the descriptor's name (member name). The third level is the type of SAS file: specify **ACCESS** for an access descriptor or **VIEW** for a view descriptor.

You can use the CREATE statement as many times as necessary in one procedure execution. That is, you can create multiple access descriptors, as well as one or more view descriptors based on these access descriptors, within the same execution of the ACCESS procedure. Or, you can create access descriptors and view descriptors in separate executions of the procedure.

Access descriptors When you create an access descriptor, you must place statements or groups of statements in a certain order after the PROC ACCESS statement and its options, as listed below:

1. CREATE statement for the access descriptor: must follow the PROC ACCESS statement.
2. Database-description statements: must follow the CREATE statement. Use either the ADBFILE or the DDM statement with the SECFILE and SYSFILE statements. Additionally with the DDM statement, use the NSS statement. The ADBFILE statement allows you to access an ADABAS file. The DDM statement accesses a view to an ADABAS file that you can use to reference the ADABAS file in NATURAL programs. In making your choice, note that the two statements use different naming conventions for ADABAS data field names.

   Information from database-description statements is stored in an access descriptor; therefore, you do not need to repeat this information when you create view descriptors. However, if no security values were entered in the access descriptor or values were provided but the SECURITY statement was set to NO, then you can use the database-description statements in a view descriptor to supply or modify them.
3. Editing statements: must follow the database-description statements. ASSIGN, CONTENT, DROP, EXTEND, FORMAT, INFORMAT, KEY, LIST, LISTINFO, LISTOCC, MVF, RENAME, RESET, and SECURITY can all be used in an access descriptor. QUIT is also an editing statement but using it terminates PROC ACCESS without creating your descriptor.
4. RUN statement: this statement is used to process the ACCESS procedure.

The order of the statements within the database-description group does not matter. For example, you could submit either the DDM= or the NSS() statement first. The order of the statements within the editing group sometimes matters; see the individual statement descriptions for any restrictions.

Note: Altering a DBMS table that has descriptor files defined on it might cause these files to be out-of-date or not valid. For example, if you re-create a table and add a new column to the table, an existing access descriptor defined on that table does not show that column; in this case the descriptor is still valid. However, if you re-create a table and delete an existing column from the table, the descriptor might not be valid. If

the deleted column is included in a view descriptor and this view is used in a SAS program, the view fails and an error message is written to the SAS log. △

View descriptors You can create view descriptors and access descriptors in the same execution of the ACCESS procedure or in separate executions.

To create a view descriptor and the access descriptor on which it is based within the *same* PROC ACCESS execution, you must place the statements or groups of statements in a particular order after the PROC ACCESS statement and its options, as listed below:

1. Create the access descriptor except omit the RUN statement.
2. CREATE statement for the view descriptor: this statement must follow the PROC ACCESS statements that created the access descriptor.
3. NSS and the password and cipher code parameters of ADBFILE, SECFILE, and SYSFILE: the ADBFILE, SECFILE, and SYSFILE statements can be specified only when SECURITY=NO or when SECURITY=YES and no values have been specified in the access descriptor referenced by this view descriptor.
4. Editing statements: SELECT and SUBSET are used only when creating view descriptors. CONTENT, FORMAT, INFORMAT, KEY, and MVF OCCURS can be specified only when ASSIGN=NO is specified in the access descriptor referenced by this view descriptor. QUIT is also an editing statement, but using it terminates PROC ACCESS without creating your descriptor.

   The order of the statements within this group usually does not matter; see the individual statement descriptions for any restrictions.
5. RUN statement: this statement is used to process the ACCESS procedure.

To create a view descriptor based on an access descriptor that was created in a *separate* PROC ACCESS step, you specify the access descriptor's name in the ACCDESC= option in the new PROC ACCESS statement. You must specify the CREATE statement before any of the editing statements for the view descriptor.

If you create only one descriptor in a PROC step, the CREATE statement and its accompanying statements are checked for errors when you submit PROC ACCESS for processing. If you create multiple descriptors in the same PROC step, each CREATE statement (and its accompanying statements) is checked for errors as it is processed.

When the RUN statement is processed, all descriptors are saved. If no errors are found, the descriptor is saved. If errors are found, error messages are written to the SAS log, and processing is terminated. After you correct the errors, resubmit your statements.

The following example creates the access descriptor ADLIB.CUSTOMER on the ADABAS CUSTOMER file using the ADBFILE statement to specify the ADABAS file.

```
/* Create access descriptor using ADABAS file */
proc access dbms=adabas;
   create adlib.customer.access;
   adbfile(number=45  password=cuspw
           cipher=cuscc dbid=1);
   sysfile(number=15  password=cuspwsys
           cipher=cusccsys  dbid=1);
   secfile(number=16  password=cuspwsec
           cipher=cusccsec  dbid=1);
   assign=yes;
   rename cu = custnum
          ph = phone
          ad = street;
   format fo = date7.;
   informat fo = date7.;
```

```
   content fo = yymmdd8.;
   mvf br occurs = 4
run;
```

The following example creates an access descriptor to the same data using the DDM statement.

```
/* Create access descriptor using NATURAL DDM */
proc access dbms=adabas;
   create adlib.customer.access;
   nss(library=sasdemo user=demo password=demopw).
   sysfile(number=15  password=cuspwsys
           cipher=cusccsys  dbid=1);
   secfile(number=16  password=cuspwsec
           cipher=cusccsec  dbid=1);
   ddm=customers;
   assign=yes;
   rename customer = custnum
         telephone = phone
         streetaddress = street;
   format firstorderdate = date7.;
   informat firstorderdate = date7.;
   content firstorderdate = yymmdd6.;
   mvf "BRANCH-OFFICE" occurs = 4
run;
```

The following example creates an access descriptor ADLIB.EMPLOY on the ADABAS EMPLOYEES file and a view descriptor VLIB.EMP1204 based on ADLIB.EMPLOY in the same PROC ACCESS step. The ADABAS file to access is referenced by a DDM.

```
/* Create access and view descriptors in
one execution */
proc access dbms=adabas;
   /* Create access descriptors */
   create adlib.employ.access;
   nss(library=sasdemo user=demo password=demopw);
   sysfile(number=15  password=cuspwsys
           cipher=cusccsys  dbid=1);
   secfile(number=16  password=cuspwsec
           cipher=cusccsec  dbid=1);
   ddm=employee;
   assign=no;
   list all;

   /* Create view descriptor  */
   create vlib.emp1204.view;
   select empid lastname hiredate salary dept
   sex    birthdate;
   format empid 6.
          salary dollar12.2
          jobcode 5.
          hiredate datetime7.
          birthdate datetime7.;
   subset where jobcode=1204;
run;
```

The following example creates a view descriptor VLIB.BDAYS from the ADLIB.EMPLOY access descriptor, which was created in a separate PROC ACCESS step.

```
/* Create view descriptors in separate execution */
proc access dbms=adabas accdesc=adlib.employ;
   create vlib.bdays.view;
   select empid lastname birthdate;
   format empid 6.
          birthdate datetime7.;
run;
```

DDM=

Indicates the NATURAL Data Definition Module (DDM) name.

Optional statement
Applies to: access descriptor
Interacts with: NSS

Syntax

DDM= *data-definition-module-name*;

Details

The DDM= statement specifies the NATURAL DDM. The name assigned to a NATURAL DDM references an ADABAS file and its data fields. Note that a DDM is often referred to as an ADABAS file, even though it is only a view of an actual ADABAS file.

The name for a NATURAL DDM can be a maximum of 32 characters. In a NATURAL DDM, data fields can be assigned a DDM external name of 3 to 32 characters. DDMs are stored in a system file that is simply another ADABAS file.

If you delete or rename a SAS/ACCESS descriptor file, you do not delete or rename the descriptor file's underlying ADABAS file or NATURAL DDM. However, changing your DDM can affect your descriptor files. See "Effects of Changing an ADABAS File or NATURAL DDM on Descriptor Files" on page 111 for more information on how changing your DDM can affect your descriptor files.

DROP

Drops a column so that it cannot be selected in a view descriptor.

Optional statement
Applies to: access descriptor
Interacts with: RESET, SELECT

Syntax

DROP *column-identifier-1 <...column-identifier-n>*;

Details

The DROP statement drops the specified column from an access descriptor. The column therefore cannot be selected by a view descriptor that is based on the access descriptor. However, the specified column in the DBMS table remains unaffected by this statement.

An editing statement, such as DROP, must follow the CREATE and database-description statements when you create an access descriptor. See "CREATE" on page 68 for more information on the order of statements.

The *column-identifier* argument can be either the column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the access descriptor. For example, to drop the third and fifth columns, submit the following statement:

```
drop 3 5;
```

If the column name contains special characters or national characters, enclose the name in quotes. You can drop as many columns as you want in one DROP statement.

To display a column that was previously dropped, specify that column name in the RESET statement. However, doing so also resets all the column's attributes (such as SAS variable name, format, and so on) to their default values.

EXTEND

Lists columns in the descriptor and gives information about them.

Optional statement

Applies to: access and view descriptors

Default ALL

Syntax

EXTEND <ALL | VIEW | *column-identifier-1* <... *column-identifier-n*>>;

Details

The EXTEND statement lists information about the informat, DB content, occurrence range, descriptor type, and BY key columns in the descriptor. For groups and periodic groups, *GROUP* or *PGROUP* is displayed, respectively.

You can use the EXTEND statement when creating an access or a view descriptor. The EXTEND information is written to your SAS log.

If you use an editing statement, such as EXTEND, it must follow the CREATE statement and the database-description statements when you create a descriptor. See "CREATE" on page 68 for more information on the order of statements.

You can specify EXTEND as many times as you want while creating a descriptor; specify EXTEND last in your PROC ACCESS code to see the completed descriptor information. Or, if you are creating multiple descriptors, specify EXTEND before the next CREATE statement to list all the information about the descriptor you are creating.

The EXTEND statement can take one of the following arguments:

ALL
: lists all the DBMS columns in the file, the positional equivalents, the two–character ADABAS names, the SAS variable informats, the database contents, occurrence ranges, descriptor types, and BY keys that are available for

the access descriptor. When you are creating an access descriptor, ***NON-DISPLAY*** appears next to the column description for any column that has been dropped. When you are creating a view descriptor, ***SELECTED*** appears next to the column description for columns that you have selected for the view.

VIEW
lists all the DBMS columns that are selected for the view descriptor, along with their positional equivalents, their two–character ADABAS names, their SAS variable informats, the database contents, occurrence ranges, descriptor types, BY keys, any subsetting clauses, and the word ***SELECTED***. Any columns that are dropped in the access descriptor are not displayed. The VIEW argument is valid only for a view descriptor.

column-identifier
lists the specified DBMS column name, its positional equivalent, its two–character ADABAS name, its SAS variable informat, the database content, occurrence range, descriptor type, BY keys that are available for the access descriptor, and whether the column has been selected or dropped. If the column name contains special characters or national characters, enclose the name in quotes.

The *column-identifier* argument can be either the column name, the positional equivalent from the LIST statement, which is the number that represents the column's place in the descriptor, or a list of column names or positions. For example, to list information about the fifth column in the descriptor, submit the following statement:

```
extend 5;
```

Or, to list information about the fifth, sixth, and eighth columns in the descriptor, submit the following statement:

```
extend 5 6 8;
```

FORMAT

Changes a SAS format for a DBMS column.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN, CONTENT, DROP, RESET

Syntax

FORMAT *column-identifier-1 <=> SAS-format-name*
<...column-identifier-n <=> SAS-format-name>;

Details

The FORMAT statement changes a SAS variable format from its default format; the default SAS variable format is based on the data type of the DBMS column. (See "ACCESS Procedure Formats and Informats" on page 98 for information about the default formats that the ACCESS Procedure assigns to your DBMS data types.)

An editing statement, such as FORMAT, must follow the CREATE statement and the database-description statements when you create a descriptor. See "CREATE" on page 68 for more information on the order of statements.

The *column-identifier* argument can be either the column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the access descriptor. For example, to associate the DATE9. format with the BIRTHDATE column and with the second column in the access descriptor, submit the following statement:

```
format 2=date9. birthdate=date9.;
```

The column-identifier is specified on the left and the SAS format is specified on the right of the expression. The equal sign (=) is optional. If the column name contains special characters or national characters, enclose the name in quotes. You can enter formats for as many columns as you want in one FORMAT statement.

You can use the FORMAT statement with a view descriptor only if the ASSIGN statement that was used when creating the access descriptor was specified with the **NO** value.

Note: You do not have to issue a SELECT statement in a view descriptor for the columns included in the FORMAT statement. The FORMAT statement selects the columns. When you use the FORMAT statement in access descriptors, the FORMAT statement reselects columns that were previously dropped with the DROP statement. △

FMT is the alias for the FORMAT statement.

INFORMAT

Changes a SAS informat for a DBMS column.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN, CONTENT, DROP, RESET

Syntax

INFORMAT *column-identifier-1 <=> SAS-format-name*
<...column-identifier-n <=> SAS-format-name>;

Details

The INFORMAT statement changes a SAS variable informat from its default informat; the default SAS variable informat is based on the data type of the DBMS column. (See "ACCESS Procedure Formats and Informats" on page 98 for information about the default informats that the ACCESS Procedure assigns to your DBMS data types.)

An editing statement, such as INFORMAT, must follow the CREATE statement and the database-description statements when you create a descriptor. See "CREATE" on page 68 for more information on the order of statements.

The *column-identifier* argument can be either the column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the access descriptor. For example, to associate the DATE9. informat with the BIRTHDATE column and with the second column in the access descriptor, submit the following statement:

```
informat 2=date9. birthdate=date9.;
```

The column-identifier is specified on the left and the SAS informat is specified on the right of the expression. The equal sign (=) is optional. If the column name contains special characters or national characters, enclose the name in quotes. You can enter informats for as many columns as you want in one INFORMAT statement.

You can use the INFORMAT statement with a view descriptor only if the ASSIGN statement that was used when creating the access descriptor was specified with the **NO** value.

Note: You do not have to issue a SELECT statement in a view descriptor for the columns included in the INFORMAT statement. The INFORMAT statement selects the columns. When you use the INFORMAT statement with access descriptors, the INFORMAT statement reselects columns that were previously dropped with the DROP statement. △

INFMT is the alias for the INFORMAT statement.

KEY

Specifies a BY key for an elementary data field that is designated as an ADABAS descriptor.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN

Default blank

Syntax

KEY<=> *column-identifier-1* <*...column-identifier-n*>;

Details

The KEY statement specifies a BY key for an elementary data field. This field must be an ADABAS descriptor.

A BY key, which is an optional set of match variables, is used only when the interface view engine must examine additional ADABAS records in order to add a new periodic group occurrence. The engine uses the BY key variables in temporary WHERE clauses that are designed to locate a record for modification. Examining the additional ADABAS records is required only if data are changed above the periodic group level from one observation to the next in a view descriptor with a selected periodic group. It is suggested that you use BY key variables even if they are not always needed.

A data field is a good candidate for a BY key variable if it uniquely identifies a logical record. The incoming values of the data fields in a BY key variable are matched to existing values in order to locate a position in which to insert new periodic groups. (A BY key variable is similar to a BY group or a BY variable in the SAS System.)

The KEY statement can have the following values:

blank	(default) indicates that the data field is not to be used as a KEY.
N	specifies that the data field is not to be used as a KEY.
Y	specifies that the data field is to be used as a KEY.

An editing statement, such as KEY, must follow the CREATE statement and the database-description statements when you create a descriptor. See "CREATE" on page 68 for more information on the order of statements.

You can use the KEY statement with a view descriptor only if the ASSIGN statement that was used when creating the acess descriptor was specified with the **NO** value.

You do not have to issue a SELECT statement in a view descriptor for the columns included in the KEY statement. The KEY statement selects the columns. When you use the KEY statement with an access descriptor, the KEY statement reselects columns that were previously dropped with the DROP statement.

LIST

Lists columns in the descriptor and gives information about them.

Optional statement

Applies to: access descriptor or view descriptor

Default: ALL

Syntax

LIST <ALL | VIEW | *column-identifier-1* <... *column-identifier-n*>>;

Details

The LIST statement lists columns in the descriptor along with information about the columns. The LIST statement can be used when creating an access descriptor or a view descriptor. The LIST information is written to your SAS log.

If you use an editing statement, such as LIST, it must follow the CREATE statement and the database-description statements when you create a descriptor. See "CREATE" on page 68 for more information on the order of statements.

You can specify LIST as many times as you want while creating a descriptor; specify LIST last in your PROC ACCESS code to see the completed descriptor information. Or, if you are creating multiple descriptors, specify LIST before the next CREATE statement to list all the information about the descriptor you are creating.

The LIST statement can take one of the following arguments:

ALL
: lists all the DBMS columns in the file, the positional equivalents, the SAS variable names, and the SAS variable formats that are available for the access descriptor. When you are creating an access descriptor, ***NON-DISPLAY*** appears next to the column description for any column that has been dropped. When you are creating a view descriptor, ***SELECTED*** appears next to the column description for columns that you have selected for the view.

VIEW
: lists all the DBMS columns that are selected for the view descriptor, along with their positional equivalents, their SAS names and formats, any subsetting clauses, and the word ***SELECTED*** . Any columns that were dropped in the access descriptor are not displayed. The VIEW argument is valid only for a view descriptor.

column-identifier
lists the specified DBMS column name, its positional equivalent, its SAS variable name and format, and whether the column has been selected or dropped. If the column name contains special characters or national characters, enclose the name in quotes.

The *column-identifier* argument can be either the column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the descriptor. For example, to list information about the fifth and eighth columns in the descriptor, submit the following statement:

```
list 5 8;
```

LISTINFO

Shows additional data field information.

Optional statement
Applies to: access descriptor or view descriptor
Default: ALL

Syntax

LISTINFO <ALL | VIEW | *column-identifier-1* <... *column-identifier-n*>>;

Details

The LISTINFO statement shows additional data field information for one or more DBMS columns in the descriptor. The LISTINFO statement can be used when creating an access or a view descriptor. The LISTINFO information is written to your SAS log.

An editing statement, such as LISTINFO, must follow the CREATE statement and the database-description statements when you create a descriptor. See "CREATE" on page 68 for more information on the order of statements.

The LISTINFO statement is especially helpful for subfields, superfields, and descriptor data fields. It shows the ADABAS level, ADABAS name, length, data type, and first-last character positions for a given DBMS column.

When you are creating an access descriptor, ***NON-DISPLAY*** appears next to the column description for any column that has been dropped. When you are creating a view descriptor, ***SELECTED*** appears next to the column description for columns that you have selected for the view.

The LISTINFO statement can take one of the following arguments:

ALL
lists all the DBMS columns in the file, the ADABAS levels, the lengths, ADABAS names, the data types, and the first-last character positions.

VIEW
lists the DBMS columns that are selected for the view descriptor, along with the ADABAS levels, ADABAS names, the lengths, the data types, and the first-last character positions. Any columns that are dropped in the access descriptor are not displayed. The VIEW argument is valid only for a view descriptor.

column-identifier
lists the specified DBMS columns, the ADABAS levels, ADABAS names, the lengths, the data types, the first-last character positions, and whether the column

has been selected or dropped. If the column name contains special characters or national characters, enclose the name in quotes.

The column-identifier argument can be either the column name, the positional equivalent from the LIST statement, which is the number that represents the column's place in the descriptor, or a list of column names or positions. For example, to list information about the fifth column in the descriptor, submit the following statement:

```
listinfo 5;
```

Or, to list information about the fifth, sixth, and eighth columns in the descriptor, submit the following statement:

```
listinfo 5 6 8;
```

LISTOCC

Lists occurrences for multiple value fields.

Optional statement

Applies to: access descriptor or view descriptor

Syntax

LISTOCC *column-identifier-1 <... column-identifier-n>*;

Details

The LISTOCC statement lists all the requested occurrences for the specified multiple-value fields along with information such as the ADABAS level, the SAS variable name, the occurrence number, the SAS variable format and informat, the DB content, the descriptor type, and whether the occurrence has been selected or dropped. The LISTOCC statement can be used when creating an access descriptor or a view descriptor. The LISTOCC information is written to your SAS log.

If you use an editing statement, such as LISTOCC, it must follow the CREATE statement and the database-description statements when you create a descriptor. See "CREATE" on page 68 for more information on the order of statements.

The LISTOCC statement takes the following argument:

column-identifier

The column-identifier argument can be either the column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the descriptor. For example, to list occurrences for the fifth column in the descriptor, submit the following statement:

```
listocc 5;
```

The column-identifier must be a multiple-value field.

MVF

Modifies the occurrences of a multiple-value field.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN

Syntax

MVF *column-identifier*

CONTENT *occurrence-1 <=> E | SAS-date-format | length*
<...occurrence-n <=> E | SAS-date-format | length>;
|
DROP *occurrence-1* <<TO> ... *occurrence-n*>;
|
FORMAT *occurrence-1 <=> SAS-format-name*
<... occurrence-n <=> SAS-format-name>;
|
INFORMAT *occurrence-1 <=> SAS-format-name*
<... occurrence-n <=> SAS-format-name>;
|
OCCURS<=> *number-of-occurrences*;
|
RENAME *occurrence-1 <=> SAS-variable-name*
< ... occurrence-n <=> SAS-variable-name>;
|
RESET *occurrence-1* <<TO> ... *occurrence-n*>;
|
SELECT *occurrence-1* <<TO> ... *occurrence-n*>;

Details

You use the MVF statement to modify values for occurrences of a multiple-value field. The MVF statement can be used when creating an access descriptor or a view descriptor.

If you use an editing statement, such as MVF, it must follow the CREATE statement and the database-description statements when you create a descriptor. See "CREATE" on page 68 for more information on the order of statements.

The MVF statement allows you to

- choose the number of occurrences by specifying a range of occurrences
- select individual occurrences or a range of occurrences
- drop individual occurrences or a range of occurrences
- reset individual occurrences or a range of occurrences
- change the format value for one or more occurrences
- change the informat value for one or more occurrences
- change the database content value for one or more occurrences
- rename the SAS variable name for one or more occurrences.

The column-identifier must be a multiple-value field, and can be the column name or the positional equivalent from the LIST statement. The occurrence argument can be the occurrence name or the occurrence number. If the column name or the occurrence name contains special characters, like `'-'`, enclose the name in quotes. The `'='` is optional for all subcommands.

You can use the LISTOCC statement to review your changes.

You do not have to issue a SELECT statement in a view descriptor for occurrences included in the CONTENT, FORMAT, INFORMAT, and RENAME subcommands. The subcommands select the columns.

The MVF statement can take one of the following subcommands:

OCCURS
allows you to specify a number of occurrences or an occurrence range. The default occurrence range is displayed as 1 191, which is the maximum number of occurrences allowed for multiple-value fields. If the value for the ASSIGN statement in an access descriptor is YES, the number of occurrences or the occurrence range cannot be changed in any view descriptor that is based on this access descriptor.

For example, if you want the BRANCH-OFFICE column in the CUSTOMER DDM to have 4 occurrences, submit the following statement:

```
mvf "BRANCH-OFFICE" occurs = 4
```

SELECT
allows you to select individual occurrences to be included in your descriptor. This subcommand is used only when defining view descriptors.

You can select one or more individual occurrences or a range of occurrences. For example, if you want to select occurrences one, two, and three of the BRANCH-OFFICE column in the CUSTOMER DDM, submit the following statement:

```
mvf "BRANCH-OFFICE" select 1 2 3;

                or

mvf "BRANCH-OFFICE" select 1 to 3;
```

DROP
allows you to drop individual occurrences from your descriptor. If you drop all occurrences of a column, the column is automatically dropped. This subcommand is used only when defining access descriptors.

You can drop one or more individual occurrences or a range of occurrences. For example, if you want to drop occurrences one, two, and three of the BRANCH-OFFICE column in the CUSTOMER DDM, submit the following statement:

```
mvf "BRANCH-OFFICE" drop 1 2 3;

                or

mvf "BRANCH-OFFICE" drop 1 to 3;
```

RESET
allows you to reset the attributes of individual occurrences. This subcommand can be used when creating an access or view descriptor. Specifying the RESET subcommand for an occurrence has the same effect on occurrence attributes as specifying the RESET statement for a column. See “RESET” on page 85 for more information.

You can reset one or more individual occurrences or a range of occurrences. For example, if you want to reset occurrences one, two, and three of the BRANCH-OFFICE column in the CUSTOMER DDM, submit the following statement:

```
mvf "BRANCH-OFFICE" reset 1 2 3;

                or

mvf "BRANCH-OFFICE" reset 1 to 3;
```

FORMAT
allows you to change the format attribute of individual occurrences. This subcommand can be used when creating access or view descriptors. However, the format attribute cannot be changed in a view descriptor when you set ASSIGN=YES.

You can change the format attribute of one or more occurrences in one FORMAT subcommand. For example, if you want to change the format attribute for occurrences nine and ten of the BRANCH-OFFICE column in the CUSTOMER DDM, submit the following statement:

```
mvf "BRANCH-OFFICE" format 9 $21.
    branch10 = $8.;
```

INFORMAT
allows you to change the informat attribute of an individual occurrence. This subcommand can be used when creating access or view descriptors. However, the informat attribute cannot be changed in a view descriptor when you set ASSIGN=YES.

You can change the informat attribute of one or more occurrences in one INFORMAT subcommand. For example, if the BRANCH-OFFICE column in the CUSTOMER DDM is a multiple-value field, and you want to change the informat attribute for occurrences nine and ten, submit the following statement:

```
mvf "BRANCH-OFFICE" informat 9 $21.
    branch10 = $8.;
```

CONTENT
allows you to change the DB content attribute of an individual occurrence. This subcommand can be used when creating access or view descriptors. Changing the DB content attribute of an occurrence has the same effect on the SAS formats and informats for ADABAS files and NATURAL DDMs as changing the DB content attribute of a column. See “CONTENT” on page 67 for more information. For example, if the FIRSTORDERDATE column in the CUSTOMER DDM is a multiple-value field, and you want to change the DB content attribute for occurrences nine and ten, submit the following statement:

```
mvf firstorderdate content 9 yymmdd6.
    branch10 = yymmdd6.;
```

RENAME
allows you to rename a SAS variable name for an individual occurrence. This subcommand can be used when creating an access or view descriptor. However, this subcommand has different effects on access and view descriptors based on the value specified in the ASSIGN statement.

If you set ASSIGN=NO in the access descriptor, the SAS variable name can be renamed. If you set ASSIGN=YES, the SAS variable name can be renamed in the access descriptor but not in the view descriptor.

You can rename the SAS variable name for one or more occurrences in one RENAME subcommand. For example, if you want to rename occurrences nine and ten of the BRANCH-OFFICE column in the CUSTOMER DDM, submit the following statement:

```
mvf "BRANCH-OFFICE" rename 9 london
    branch10 = tokyo;
```

You can use the LISTOCC statement to review your changes.

NSS

Specifies the NATURAL SECURITY options in the access descriptor.

Optional statement

Applies to: access descriptor

Interacts with: DDM and SECURITY

Syntax

NSS (LIBRARY | LIB = *library-identifier*
USER = *user-identifier*
PASSWORD | PW = *Natural-Security-password*);

Details

Note: This statement is used only when a DDM is specified; otherwise, it is ignored. △

The NSS statement specifies NATURAL SECURITY options, including a library identifier, user identifier, and a password.

If you specify **YES** for the SECURITY statement in an access descriptor, values declared for Library, User, and Password cannot be changed in a subsequent view descriptor based on the access descriptor.

Library
is an eight-character library identifier. The first character must be alphabetic. The library identifier is the same as the application identifier in SAS/ACCESS Interface to ADABAS, Version 6.

User
is an eight-character user identifier.

Password
is an eight-character ADABAS password. The value is written to the access descriptor in encrypted form.

QUIT

Terminates the procedure.

Optional statement

Applies to: access descriptor or view descriptor

Syntax

QUIT;

Details

The QUIT statement terminates the ACCESS procedure without any further descriptor creation.

EXIT is the alias for the QUIT statement.

RENAME

Modifies the SAS variable name.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN, RESET

Syntax

RENAME *column-identifier-1 <=> SAS-variable-name*
<...column-identifier-n <=> SAS-variable-name>;

Details

The RENAME statement enters or modifies the SAS variable name that is associated with a DBMS column. The RENAME statement can be used when creating an access descriptor or a view descriptor.

An editing statement, such as RENAME, must follow the CREATE statement and the database-description statements when you create a descriptor. See "CREATE" on page 68 for more information on the order of statements.

Two factors affect the use of the RENAME statement: whether you specify the ASSIGN statement when you are creating an access descriptor, and the kind of descriptor you are creating.

- If you omit the ASSIGN statement or specify it with a **NO** value, the renamed SAS variable names that you specify in the access descriptor are retained throughout the access descriptor and any view descriptor that is based on that access descriptor. For example, if you rename the CUSTOMER column to CUSTNUM when you create an access descriptor, that column continues to be named CUSTNUM when you select it in a view descriptor unless a RESET statement or another RENAME statement is specified.

  When creating a view descriptor that is based on this access descriptor, you can specify the RESET statement or another RENAME statement to rename the variable again, but the new name applies only in that view. When you create other view descriptors, the SAS variable names are derived from the access descriptor.
- If you specify the **YES** value in the ASSIGN statement, you can use the RENAME statement to change SAS variable names only while creating a specific access descriptor. As described earlier in the ASSIGN statement, SAS variable names that are saved in an access descriptor are always used when creating view descriptors that are based on it.

  Renamed SAS variable names only apply to the current access descriptor that is being created. The default SAS variable names will be used for any subsequent access descriptors that are created in the same ACCESS procedure execution.

The *column-identifier* argument can be either the DBMS column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the descriptor. For example, to rename the SAS variable names that are associated with the seventh column and the nine-character FIRSTNAME column in a descriptor, submit the following statement:

```
rename 7 birthdy  firstname=fname;
```

The DBMS column name (or positional equivalent) is specified on the left side of the expression, with the SAS variable name on the right side. The equal sign (=) is optional. If the column name contains special characters or national characters, enclose the name in quotes. You can rename as many columns as you want in one RENAME statement.

When you are creating a view descriptor, the RENAME statement automatically selects the renamed column for the view. That is, if you rename the SAS variable associated with a DBMS column, you do not have to issue a SELECT statement for that column.

RESET

Resets DBMS columns to their default settings.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN, CONTENT, DROP, FORMAT, INFORMAT, KEY, MVF, RENAME, SELECT

Syntax

RESET <ALL | *column-identifier-1* <... *column-identifier-n*>>;

Details

The RESET statement resets either the attributes of all the columns or the attributes of the specified columns to their default values. The RESET statement can be used when creating an access descriptor or a view descriptor. However, this statement has different effects on access and view descriptors, as described below.

If you use an editing statement, such as RESET, it must follow the CREATE statement and the database-description statements when you create a descriptor. See “CREATE” on page 68 for more information on the order of statements.

Access descriptors

When you create an access descriptor, the default setting for a SAS variable name is a blank. However, if you have previously entered or modified any of the SAS variable names, the RESET statement resets the modified names to the default names that are generated by the ACCESS procedure. How the default SAS variable names are set depends on whether you included the ASSIGN statement. If you omitted ASSIGN or set it to **NO**, the default names are blank. If you set **ASSIGN=YES**, the default names are the first eight characters of each DBMS column name.

The current SAS variable format and informat are reset to the default SAS format and informat, which was determined from the column's data type. The current DB content, occurrence range, and BY key are also reset to the default values. Any columns that were previously dropped, that are specified in the RESET command, become available; they can be selected in view descriptors that are based on this access descriptor.

View descriptors

When you create a view descriptor, the RESET statement clears any columns that were included in the SELECT statement (that is, it "de-selects" the columns).

When creating the view descriptor, if you reset a SAS variable and then select it again within the same procedure execution, the SAS variable name, format, informat, database content, occurrence range, and BY key are reset to their default values, (the SAS name is generated from the DBMS column name, and the format and informat values are generated from the data type). This applies only if you have omitted the ASSIGN statement or set the value to **NO** when you created the access descriptor on which the view descriptor is based. If you specified **ASSIGN=YES** when you created the access descriptor, the RESET statement has no effect on the view descriptor.

The RESET statement can take one of the following arguments:

ALL
for access descriptors, resets all the DBMS columns that have been defined to their default names and format settings and reselects any dropped columns.

For view descriptors, ALL resets all the columns that have been selected, so that no columns are selected for the view; you can then use the SELECT statement to select new columns.

column-identifier
can be either the DBMS column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the access descriptor. For example, to reset the third column, submit the following statement:

```
reset 3;
```

If the column name contains special characters or national characters, enclose the name in quotes. You can reset as many columns as you want in one RESET statement, or use the ALL option to reset all the columns.

SECFILE

Specifies parameters for the NATURAL SECURITY system file.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: SECURITY

Syntax

SECFILE (NUMBER | NUM = *Natural-Security-file-number*
PASSWORD | PW = *Adabas-password*
CIPHER|CC = *Adabas-cipher-code*
DBID = *Adabas-database-identifier*);

Details The SECFILE statement allows you to specify the ADABAS file number, password, cipher code, and database identifier for the NATURAL SECURITY system file.

If you specified SECURITY=YES in the access descriptor, you cannot change the values for the password and the cipher code in the view descriptor based on this access descriptor. However, if no values were specified in the parent access descriptor, then the

values can be entered in the view descriptor, even when the SECURITY=YES statement has been issued.

Note that you can associate a password, cipher code, and database identifier with an ADABAS file number, system file, and security file.

Number
is the ADABAS file number of the NATURAL SECURITY system file. This file contains the NATURAL SECURITY library identifier, user identifier, and passwords.

Password
is an ADABAS password, which provides security protection at the file or data-field level, or on the basis of a value at the logical-record level. The value is written to the access descriptor in encrypted form.

Cipher Code
is an ADABAS cipher code, which is a numeric code for ciphering and deciphering data into and from an ADABAS file. The value is written to the access descriptor in encrypted form.

DBID
is the ADABAS database identifier (number) to be accessed. The database identifier is a numerical value from 1 to 255 that is assigned to each ADABAS database.

SECURITY

Controls the enforcement of security specifications.

Optional statement

Applies to: access descriptor

Interacts with: ADBFILE, SECFILE, SYSFILE

Default: NO

Syntax

SECURITY<=> YES | NO | Y | N;

Details

The SECURITY statement has the default value **NO**. Its value controls the enforcement of security specifications when you later create view descriptors based on this access descriptor.

With a value of **NO**, when you create view descriptors based on this access descriptor, you will be able to modify specified values for ADABAS passwords and cipher codes.

With a value of **YES**, when you create view descriptors based on this access descriptor, you will not be able to modify specified values forADABAS passwords and cipher codes. However, any values that are not specified in the access descriptor can be specified in a view descriptor or with a data set option.

SELECT

Selects DBMS columns for the view descriptor.

Required statement

Applies to: view descriptor

Interacts with: RESET

Syntax

SELECT ALL | *column-identifier-1* <...*column-identifier-n*>;

Details

The SELECT statement specifies which DBMS columns in the access descriptor to include in the view descriptor. This is a required statement and is used only when defining view descriptors.

If you use an editing statement, such as SELECT, it must follow the CREATE statement when you create a view descriptor. See "CREATE" on page 68 for more information on the order of statements.

The SELECT statement can take one of the following arguments:

ALL
includes in the view descriptor all the columns that were defined in the access descriptor excluding dropped columns.

column-identifier
can be either the DBMS column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the access descriptor on which the view is based. For example, to select the first three columns, submit the following statement:

```
select 1 2 3;
```

If the column name contains special characters or national characters, enclose the name in quotes. You can select as many columns as you want in one SELECT statement.

SELECT statements are cumulative within the same view creation. That is, if you submit the following two SELECT statements, columns 1, 5, and 6 are selected, not just columns 5 and 6:

```
select 1;
select 5 6;
```

To clear all your current selections when creating a view descriptor, use the **RESET ALL** statement; you can then use another SELECT statement to select new columns.

SUBSET

Adds or modifies selection criteria for a view descriptor.

Optional statement

Applies to: view descriptor

Syntax

SUBSET <*selection-criteria*>;

Details

You use the SUBSET statement to specify selection criteria when you create a view descriptor. This statement is optional; if you omit it, the view retrieves all the data (that is, all the rows) in the DBMS table.

An editing statement, such as SUBSET, must follow the CREATE statement when you create a view descriptor. See "CREATE" on page 68 for more information on the order of statements.

The selection-criteria argument can be either a WHERE clause or a SORT clause. For more information on the WHERE clause, see "WHERE Clause in a View Descriptor" on page 92. For more information on the SORT clause, see "SORT Clause in a View Descriptor" on page 97. You can use either SAS variable names or DBMS column names, in your selection criteria. Specify your WHERE clause and SORT clause by using separate SUBSET statements. For example, you can submit the following SUBSET statements:

```
subset where jobcode = 1204;
subset sort lastname;
```

The SAS System does not check the SUBSET statement for errors. The statement is verified and validated only when the view descriptor is used in a SAS program.

To delete the selection criteria, submit a SUBSET statement without any arguments.

SYSFILE

Specifies parameters for the system file containing DDMs.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: SECURITY

Syntax

SYSFILE (NUMBER | NUM = *Adabas-system-file-number*
PASSWORD | PW = *Adabas-password*
CIPHER|CC = *Adabas-cipher-code*
DBID = *Adabas-database-identifier*);

Details

The SYSFILE statement allows you to specify the ADABAS file number, password, cipher code, and database identifier for the system file containing DDMs.

If you specified SECURITY=YES in the access descriptor, you cannot change the values for the password and cipher code in the view descriptor. However, if no values

were entered in the access descriptor, you can enter them in the view descriptor, even if the SECURITY=YES statement has been issued.

Note that you can associate a password, cipher code, and database identifier with an ADABAS file number, system file, and security file.

Number
is the ADABAS file number of the system file containing DDMs.

Password
is an ADABAS password, which provides security protection at the file or data-field level, or on the basis of a value at the logical-record level. The value is written to the access descriptor in encrypted form.

Cipher Code
is an ADABAS cipher code, which is a numeric code for ciphering and deciphering data into and from an ADABAS file. The value is written to the access descriptor in encrypted form.

DBID
is the ADABAS database identifier (number) to be accessed. The database identifier is a numerical value from 1 to 255 that is assigned to each ADABAS database.

UPDATE

Updates a SAS/ACCESS descriptor file.

Optional statement

Applies to: access descriptor or view descriptor

Syntax

UPDATE *libref.member-name.*ACCESS | VIEW
<*password-level=SAS-password*>;

Details The UPDATE statement identifies an existing access descriptor or view descriptor that you want to update. The descriptor can exist in either a temporary (WORK) or permanent SAS data library. If the descriptor has been protected with a SAS password that prohibits editing of the ACCESS or VIEW descriptor, then the password must be specified on the UPDATE statement.

Note: It is recommended that you re-create (or overwrite) your descriptors rather than update them. SAS does not validate updated descriptors. If you create an error while updating a descriptor, you will not know of it until you use the descriptor in a SAS procedure such as PROC PRINT. △

To update a descriptor, use its three-level name. The first level identifies the libref of the SAS data library where you stored the descriptor. The second level is the descriptor's name (member name). The third level is the type of SAS file: ACCESS or VIEW.

You can use the UPDATE statement as many times as necessary in one procedure execution. That is, you can update multiple access descriptors, as well as one or more view descriptors based on these access descriptors, within the same execution of the ACCESS procedure. Or, you can update access descriptors and view descriptors in separate executions of the procedure.

You can use the CREATE statement and the UPDATE statement in the same procedure execution.

If you update only one descriptor in a procedure execution, the UPDATE and its accompanying statements are checked for errors when you submit the procedure for processing. If you update multiple descriptors in the same procedure execution, each UPDATE statement (and its accompanying statements) is checked for errors as it is processed. In either case, the UPDATE statement must be the first statement after the PROC ACCESS statement (Note: The ACCDESC= parameter cannot be specified on the PROC ACCESS statement).

When the RUN statement is processed, all descriptors are saved. If errors are found, error messages are written to the SAS log, and processing is terminated. After you correct the errors, resubmit your statements.

The following statements are not supported when using the UPDATE statement: ASSIGN, RESET, SECURITY, SELECT, and MVF subcommands RESET and SELECT.

Note: You cannot create a view descriptor after you have updated a view descriptor in the same procedure execution. You can create a view descriptor after updating or creating an access descriptor or after creating a view descriptor. △

The following example updates the access descriptor MYLIB.ORDER on the ADABAS file ORDER. In this example, the column names are changed and formats are added.

```
proc access dbms=adabas;
   update mylib.order.access;
   rename ordernum ord_num
          fabriccharges fabrics;
   format firstorderdate date7.;
   informat firstorderdate date7.;
   content firstorderdate yymmdd6.;
 run;
```

The following example updates an access descriptor ADLIB.EMPLOY on the ADABAS file EMPLOYEE and then re-creates a view descriptor VLIB.EMP1204, which was based on ADLIB.EMPLOY. The original access descriptor included all of the columns in the file. Here, the salary and birthdate columns are dropped from the access descriptor so that users cannot see this data. Because RESET is not supported when UPDATE is used, the view descriptor VLIB.EMP1204 must be re-created in order to omit the salary and birthdate columns.

```
proc access dbms=adabas;
   /* update access descriptor  */
   update adlib.employ.access;
   drop salary birthdate;
   list all;

   /* re-create view descriptor */
   create vlib.emp1204.view;
   select empid hiredate dept jobcode sex
          lastname firstname middlename phone;
   format empid 6.
          hiredate date7.;
   subset where jobcode=1204;
run;
```

The following example updates a view descriptor VLIB.BDAYS from the ADLIB.EMPLOY access descriptor, which was created in a separate procedure execution. In this example, the WHERE clause replaces the WHERE clause that was specified in the original view descriptor.

```
proc access dbms=adabas
   update vlib.bdays.view;
   subset;
   subset where empid GT 212916;
run;
```

WHERE Clause in a View Descriptor

You can use a WHERE clause in a view descriptor to select specific ADABAS records.

View WHERE Clause Syntax

A view WHERE clause consists of the SUBSET and WHERE (or WH) keywords, followed by one or more conditions that specify criteria for selecting records. A condition has one of the following forms:

```
field-name<(occurrence)> operator value
field-name<(occurrence)> range-operator
   low-value * high-value
```

The user-supplied elements of the WHERE clause conditions are described below:

field-name
is the ADABAS name of the data field or corresponding SAS variable name for which you are specifying criteria. This data field must be selected in the view descriptor. (The procedure will assume that any name in a condition is a SAS name. If it is not, the procedure will treat it as an ADABAS name.) If the field's ADABAS name is not unique within a NATURAL DDM, you must specify its external name.

A referenced data field must be an ADABAS descriptor field in the following situations:

- The view WHERE clause contains more than one condition.
- The view WHERE clause uses the SPANS or NE operator.
- You are also specifying a view SORT clause.
- You are also planning to issue a SAS BY statement or a SAS ORDER BY clause in a SAS program that references a view descriptor containing a view WHERE clause.
- You are also planning to issue a SAS WHERE clause in a SAS program that references a view descriptor containing a view WHERE clause.

(*occurrence*)
is a numeric value from 1 to 99 identifying the *n*th occurrence of a periodic group. You must use parentheses around the number. This is an optional value. If you do not specify an occurrence number, all occurrences are selected.

operator
can be one of the following comparison and logical operators:

= or EQ	equal to
> or GT	greater than
< or LT	less than
!= or ¬= or NE	not equal to

≥ or GE or GTE greater than or equal to

≤ or LE or LTE less than or equal to

range-operator
can be one of the following operators:

= or EQ or SPANS within the range (inclusive)

value or *high-value* or *low-value*
is a valid value for the data field.

View WHERE Clause Examples

This section gives brief examples using the WHERE clause and explains what each example does.

Specifying Conditions with the SPANS Operator

When comparing low and high values, the asterisk is required. For example, the following WHERE clause selects those employees with employee numbers between 2300 and 2400:

```
subset where personnel-number spans 2300 * 2400
```

The following WHERE clause selects those employees with last names up through Smith:

```
subset where name spans 'A' * 'Smith'
```

Specifying Expressions

You can combine conditions to form expressions. Two conditions can be joined with OR (|) or AND (&). Since expressions within parentheses are processed before those outside, use parentheses to have the OR processed before the AND.

```
subset where cost = .50 & (type = ansi12 |
    class = sorry)
```

The following WHERE clause selects all records where AVAIL is Y or W:

```
subset where avail eq y | avail eq w
```

The next WHERE clause selects all records where PART is 9846 and ON-HAND is greater than 1,000:

```
subset where part = 9846 & on-hand > 1000
```

Specifying Values in Character Fields

For character fields, you can use quoted or unquoted strings. Any value entered within quotes is left as is; all unquoted values are uppercased and redundant blanks are removed. For example, the following clause extracts data for SMITH:

```
subset where lastname = Smith
```

The next example extracts data for Smith:

```
subset where lastname = 'Smith'
```

The next WHERE clause selects all records where CITY is TRUTH OR CONSEQUENCES or STZIP is NM 87901. Notice in the first condition that quotes prevent OR from being used as an operator. In the second condition, they prevent the extra space between NM and 87901 from being removed.

```
subset where city = 'TRUTH OR CONSEQUENCES' |
    stzip = 'NM  87901'
```

The following example selects all records where SHOP is Joe's Garage. Because the value is enclosed in quotes, the two consecutive single quotes are treated as one.

```
subset where shop = 'Joe''s Garage'
```

You can also use double quotes, for example,

```
subset where shop = "Joe's Garage"
```

Specifying Numeric Format Values

For numeric values, use decimal or scientific notation. For example,

```
subset where horsepower = 2.5
```

Specifying Dates

Numeric values representing dates in an ADABAS file are not automatically converted to SAS date values. They are simply treated as numbers. For example, 103098 is considered less than 113188.

However, the ACCESS procedure provides you the ability to specify a SAS date format with the CONTENT statement. Then, numeric values are converted to SAS dates. To reference them in a view WHERE clause, use informat representation (without the 'D at the end as in the SAS System). See "CONTENT" on page 67 for more information on specifying a SAS date format with the CONTENT statement.

Specifying Values in Superdescriptor Fields

A superdescriptor field is treated as if it has an alphanumeric (character) ADABAS standard format unless all of the parent fields from which it is derived have a binary (numeric) format.

When you enter a value for a numeric superdescriptor or an alphanumeric superdescriptor where one or more of its parent fields have a numeric format, the value must be in character hexadecimal format because many data types and from-to specifications can be contained in one superdescriptor value. When you enter a value for a character superdescriptor, the value must be entered as character data.

Note: By assigning a SAS format of HEX*w*. to superdescriptors that are derived from one or more numeric fields in a view descriptor, you can see the internal hexadecimal values. You can then use these values as a guide for entering like values in the WHERE clause. △

For example, the NATURAL DDM named CUSTOMERS has the character superdescriptor field STATE-ZIPLAST2, which is defined as

```
'SP=ST(1,2),ZI(1,2)'
```

The two data fields that make up STATE-ZIPLAST2 are defined as

```
DDM Name     ADABAS ID     ADABAS TYPE     LENGTH
--------     ---------     -----------     ------
STATE           ST              A             2
ZIPCODE         ZI              U             5
```

If you want to select the value **TX** from the data field STATE and the value 78701 from the data field ZIPCODE, the view WHERE clause would be as follows:

```
subset where state_zi = E3E7F0F1
```

The comparable SAS WHERE clause would be

```
where state_zi = 'E3E7F0F1'x
```

F0F1 is the hexadecimal internal representation of a positive zoned decimal value of 01. If ZIPCODE were defined as packed and the from-to specification were the same, the hexadecimal representation 001F would represent the value 01. Similarly, 0001 would be the correct representation for either binary or fixed. A sign (+ or -) must also be entered according to type and ADABAS requirements.

Suppose you want to access a character superdescriptor field named DEPT-PERSON, which is defined as

```
'S2=DP(1,6),LN(1,18)'
```

The two data fields that make up DEPT-PERSON are defined as

```
DDM Name     ADABAS ID     ADABAS TYPE     LENGTH
--------     ---------     -----------     ------
  DEPT          DP              A             6
  LASTNAME      LN              A             18
```

If you want to select the value **TECH01** from the data field DEPT and the value **BOYER** from the data field LASTNAME, the view WHERE clause would be as follows. (Note that unquoted values in the view WHERE clause are uppercased.)

```
subset where dept-person = tech01boyer
```

A comparable SAS WHERE clause would be

```
where dept-person = 'TECH01BOYER'
```

Specifying Values in Subdescriptor Fields

Subdescriptors take the ADABAS type of their parent and the length of their from-to specification. Unlike superdescriptors, subdescriptor values consist of only one data type.

For example, the NATURAL DDM named CUSTOMERS has the numeric subdescriptor field ZIPLAST, which is defined as

```
'SB=ZI(1,2)'
```

The data field that ZIPLAST is based on is defined as

```
DDM Name     ADABAS ID     ADABAS TYPE     LENGTH
--------     ---------     -----------     ------
ZIPCODE         ZI              U             5
```

If you want to select the values 78701, 82701, and 48301, the view WHERE clause and the SAS WHERE clause would be as follows.

View WHERE clause:

```
subset where ziplast2 = 01
```

SAS WHERE clause:

```
where ziplast2 = 01
```

Now suppose you want to access a character subdescriptor field named DEPT-CODE, which is defined as

```
'DC=DP(1,4)'
```

The data field that DEPT-CODE is based on is defined as

```
DDM Name    ADABAS ID    ADABAS TYPE    LENGTH
--------    ---------    -----------    ------
  DEPT          DP             A            6
```

If you want to select the values **TECH01**, **TECH04**, and **TECH23**, the view WHERE clause would be

```
subset where dept-code = tech
```

The comparable SAS WHERE clause would be

```
where dept-code = 'TECH'
```

Specifying Values in Multiple-Value Fields

If the field name refers to a multiple-value field, all values for the field are compared with the value that you specify. For example, if CARD is a multiple-value field, the following view WHERE clause selects all records where any one of the values of CARD is VISA.

```
subset where card eq visa
```

Note that in a SAS WHERE clause, you cannot specify a value for a multiple-value field; however, in a SAS WHERE clause, you can specify an occurrence, which you cannot do in a view WHERE clause.

For more information and examples of using multiple-value fields in selection criteria, see “Using Multiple-Value Fields in Selection Criteria” on page 127.

Specifying Values in Periodic Group Fields

If the field is in a periodic group, use *field-name(occurrence)* to identify the field in the *n*th occurrence of the group. For example, the following WHERE clause selects all records where PHONE is 234-9876 in the second occurrence of the periodic group containing PHONE.

```
subset where phone(2) eq 234-9876
```

Note that the 2 after PHONE refers to the second occurrence of its parent periodic group and not to the second occurrence of PHONE.

If you do not specify an occurrence number, all occurrences are checked. For example, the following WHERE clause selects all records where PHONE is 234-9876 in any occurrence of the periodic group containing PHONE.

```
subset where phone eq 234-9876
```

For more information and examples of using periodic group fields in selection criteria, see “Using Multiple-Value Fields in Selection Criteria” on page 127.

SORT Clause in a View Descriptor

When you define a view descriptor, you can also include a SORT clause to specify data order. You can reference only the data fields selected for the view descriptor, and the data fields must be descriptors; that is, they must have indexes. Without a SORT clause or a SAS BY statement, the data order is determined by ADABAS.

A SAS BY statement automatically issues a SORT clause to ADABAS. If a view descriptor already contains a SORT clause, the BY statement overrides the sort for that program. An exception is when the SAS procedure includes the NOTSORTED option. Then, the SAS BY statement is ignored, and the view descriptor SORT clause is used; a message is written to the log when NOTSORTED causes a SAS BY statement to be ignored.

View SORT Clause Syntax

The syntax for the SORT clause is
SUBSET SORT *field-name* <,*field-name*> <,*field-name*> <*option*>
The elements of the SORT clause are described below.

field-name
is the name of an ADABAS data field or its corresponding SAS variable name to sort by. The data field must be an ADABAS descriptor; that is, it must be a key data field. You can use the data field's ADABAS field name or its DDM name.

You can specify up to three data fields; optionally, you can separate them with commas. If you specify more than one field name, the values are ordered by the first named field, then the second, and so on.

option
is one of the following, which applies to all specified field names. That is, you cannot specify an option for one field name and a different option for another field name.

<ASCENDING|ASCENDISN|DESCENDING>

ASCENDING
indicates the sort is to be in ascending order (low-to-high). For example, A, B, C, D or 1, 2, 3 4. The default is ASCENDING.

ASCENDISN
indicates the sort is to be in ascending ISN (internal sequence number) order. Each logical record in an ADABAS file has an assigned ISN for identification. If you specify ASCENDISN, you cannot specify a data field name.

DESCENDING
indicates the sort is to be in descending order (high-to-low). For example, Z, Y, X, W or 9, 8, 7 6.

SORT Clause Examples

The following SORT clause causes the ADABAS values to be presented in ascending order. Based on the data fields included in the VLIB.USACUST view descriptor, the logical records are presented first by the values in the data field CUSTOMER, then by the values in data field ZIPCODE, and then by the values in the data field FIRSTORDERDATE.

```
subset sort customer, zipcode, firstorderdate
```

The following SORT clause causes logical records that are accessed by the VLIB.CUSPHON view descriptor to be presented in descending order based on the values in the NAME data field:

```
subset sort name descending
```

Creating and Using View Descriptors Efficiently

When creating and using view descriptors, follow these guidelines to minimize ADABAS processing and your operating system resources and to reduce the time ADABAS takes to access data.

- Specify selection criteria to subset the number of logical records ADABAS returns to the SAS System.
- Write selection criteria that allow ADABAS to use inverted lists when possible. This applies whether you specify the selection criteria as part of the view descriptor or in a SAS program. This is especially important when accessing a large ADABAS file.

  When ADABAS cannot use an inverted list, it sequentially scans the entire file. You cannot guarantee that ADABAS will use an inverted list to process a condition on a descriptor data field, but you can write selection criteria that allow ADABAS to use available inverted lists effectively.
- Select only the data fields your program needs. Selecting unnecessary data fields adds extra processing time and requires more memory.
- Use a BY statement to specify the order in which logical records are presented to the SAS System only if the SAS System needs the data in a particular order for subsequent processing. You can use ADABAS descriptor data fields only.

  As an alternative to using a BY statement, which consumes CPU time each time you access the ADABAS file, you could use the SORT procedure with the OUT= option to create a sorted SAS data file. In this case, the SAS System, not ADABAS, does the sorting. This is a better approach for data that you want to use many times.
- If a view descriptor describes a large amount of ADABAS data and you will use the view descriptor often, it might be more efficient to extract the data and place them in a SAS data file. See “Performance Considerations” on page 36 for more information on when it is best to extract data.
- If you don't need all occurrences of multiple-value fields, limit the number of occurrences with the MVF statement.
- If you reference data fields in selection criteria that are not ADABAS descriptors, it is generally more efficient to put those conditions in a SAS WHERE clause, not in the view descriptor WHERE clause.
- To optimize WHERE clause processing, the ADABAS interface view engine uses the ADABAS L3 command when possible. However, a number of restrictions must be satisfied before the L3 command can be used. For these restrictions, see “How the SAS/ACCESS Interface to ADABAS Works” on page 106.

ACCESS Procedure Formats and Informats

When you create SAS/ACCESS descriptor files from ADABAS data, the ACCESS procedure converts data field types and lengths to default SAS System variable formats and informats.

The following summary information will help you understand the data conversion.

- The ADABAS interface view engine uses ADABAS standard length and type for reading and updating ADABAS data (except for variable-length fields and DB Content overrides). NATURAL DDMs have no effect other than to use DDM length and decimals to set SAS formats.
- Length and decimal points specified by DDMs might conflict with the ADABAS file definition (for example, not big enough, too big, and so on). If so, the ADABAS standard length is used to set default SAS formats.
- Packed, unpacked, and binary types can hold very large numeric data values. The SAS System can maintain precision up to sixteen digits. Unpacked fields larger than sixteen bytes are converted to the character hexadecimal type upon which no numeric operations can occur. Therefore, precision is not a problem. For large packed and binary fields, however, you must be aware that precision can be lost when data values exceed sixteen digits.
- If the standard length is 0 (that is, if the data field has a variable length), the ACCESS procedure chooses a default length.
  - The default length for an alphanumeric is 20.
  - The default length for a numeric is the maximum length before assuming a character hexadecimal type. Packed is 15 bytes (29 digits and a sign), unpacked is 16 bytes (16 digits and a sign), binary is 8 bytes, fixed is 4 bytes, and float is 8 bytes.
- Superdescriptors and subfields are given an ADABAS type of character unless all of the parent fields are numeric. Then, they are given an ADABAS type of binary. Their length is calculated by totaling the number of bytes in the individual parent's from-to specification. If the length of a binary superdescriptor or subdescriptor is greater than 8, the SAS format is changed from numeric to character hexadecimal.
- Subdescriptors and subfields take the type of their parent and the length of their from-to specification.
- Phonetic descriptors are alphanumeric and use the length of the phonetic parent. Any retrieval of a phonetic descriptor is actually retrieval of its parent.
- If ADABAS data fall outside the valid SAS data ranges, you will get an error message in the SAS log when you try to read the data. For example, an ADABAS date might not fall in the valid SAS date range.

The following table shows the default SAS System variable formats and informats that the ACCESS procedure assigns to each ADABAS data type in an ADABAS file.

ADABAS Type	Description	Standard Length in Bytes	SAS Format and Informat
A	alphanumeric	<=200	$*ADBLEN*.
		>200	$200
B	binary	< = 4	(2 x *ADBLEN*) + 1
	(unsigned)	> 4 and < =8	(2 x *ADBLEN*).
		> 8 and < =100	$HEX(2 x *ADBLEN*) .
		> 100	$HEX200.
F	fixed (signed)		8.

ADABAS Type	Description	Standard Length in Bytes	SAS Format and Informat
G	floating point (signed)		BEST12.
P	packed decimal (signed)		(2 x *ADBLEN* + 1).
U	unpacked decimal	< = 16	(*ADBLEN* + 1).
	(zoned decimal)	> 16	$HEX(2 x *ADBLEN*).
	(signed)		

The following information applies to this table:

- *ADBLEN* = ADABAS standard length (in bytes). If the standard length equals 0, then the interface view engine sets the length based on the data type, as follows: A=20, B=8, F=4, G=8, P=15, and U=16.
- Binary data that are
  - < = 4 bytes are treated as signed numbers
  - < = 8 bytes and > 4 bytes are treated as positive (unsigned) numbers
  - > 8 bytes are treated as character hexadecimal data.
- Numeric values greater than 16 displayable digits can lose precision.

The following table shows the default SAS System variable formats and informats that the ACCESS procedure assigns to each ADABAS data type in a NATURAL DDM.

ADABAS Type	Description	Standard Length in Bytes	SAS Format and Informat
A	alphanumeric	< = 200	$*DDMLEN*.
		> 200	$200.
B	binary (unsigned)	< = 4	(*DDMLEN* + *DECPT* + *SIGNPT*) .
		> 4 and < = 8	(*DDMLEN* +*DECPT*) .
		> 8 and < = 100	$HEX(2 x ADBLEN).
		> 100	$HEX200.
F	fixed (signed)		(*DDMLEN* + *DECPT* + *SIGNPT*) .
G	floating point (signed)		BEST12.
P	packed decimal (signed)		(*DDMLEN* + *DDMDEC* + *DECPT* + *SIGNPT*) . *DDMDEC*.
U	unpacked decimal (zoned decimal) (signed)	< = 16	(*DDMLEN* + *DDMDEC* + *DECPT* + *SIGNPT*) . *DDMDEC*.
		> 16	$HEX(2 x *ADBLEN*).

The following information applies to this table:

- *DDMLEN* = DDM digits to the left of the decimal point.
- *DDMDEC* = DDM digits to the right of the decimal point.
- *ADBLEN* = ADABAS standard length in bytes. If the standard length equals 0, then the interface view engine sets the length based on the data type, as follows: A=20, B=8, F=4, G=8, P=15, and U=16.
- *DECPT* = 1 when DDM digits to the right of the decimal point are greater than 0.

- *DECPT* = 0 when DDM digits to the right of decimal point are equal to 0.
- *SIGNPT* = 1 when numeric type is signed data (fixed, float, packed, unpacked, and binary ≤4).
- *SIGNPT* = 0 when numeric type is unsigned data (binary > 4 and ≤ l8).
- Binary data that are
  - ≤ 4 bytes are treated as signed numbers
  - ≤ 8 bytes and > 4 bytes are treated as positive (unsigned) numbers
  - > 8 bytes are treated as character hexadecimal data.
- Numeric values greater than 16 displayable digits can lose precision.

Effects of the SAS/ACCESS Interface on ADABAS Data

When you access ADABAS data through the SAS/ACCESS interface, the interface view engine maps the ADABAS data into SAS observations.

- Multiple-value field occurrences are mapped to multiple SAS variables. For example, if the ADABAS data have a multiple-value field named JOBTITLE with two occurrences, the resulting SAS variables would be JOBTITL1 and JOBTITL2.
- Periodic group occurrences are mapped to multiple SAS observations. For example, if the ADABAS data have a periodic group field named EDUCATION consisting of data fields COLLEGE, DEGREE, and YEAR, there would be one observation for COLLEGE, DEGREE, and YEAR for each periodic group occurrence.

When you create SAS/ACCESS descriptor files for ADABAS data, you need to be aware of how some data fields are affected by the ACCESS procedure and how you can use them as variables in SAS programs.

- When you create a SAS/ACCESS descriptor file for ADABAS data, the ACCESS procedure automatically creates a SAS variable named ISN. This variable gives you access to the ISNs (internal sequence numbers) for all the ADABAS logical records.
- Selecting either a *subdescriptor* or a *superdescriptor* data field creates a SAS variable for the data field. The variable can be retrieved and used in a WHERE clause; however, the variable cannot be updated.
- Selecting a *phonetic descriptor* data field creates a SAS variable for that phonetic descriptor. The values of the data field for which the phonetic descriptor is defined are retrieved, and the phonetic descriptor can be used in a WHERE clause. However, this variable cannot be updated.

  If you use a variable for a phonetic descriptor in a SAS WHERE clause, the interface view engine must be able to process the entire SAS WHERE clause.
- For a *multiple-value* data field, the ACCESS procedure creates SAS variables that reference individual occurrences and a SAS variable that references all occurrences to perform special WHERE clause queries. For example, in the NATURAL DDM named CUSTOMERS, the BRANCH-OFFICE data field is a multiple-value data field with four occurrences. The ACCESS procedure creates SAS variables named BRANCH_1, BRANCH_2, and so on, and a SAS variable named BR_ANY. For more information and examples, see “Using Multiple-Value Fields in Selection Criteria” on page 127.
- For a *periodic group* data field, the ACCESS procedure creates a SAS variable for the occurrence number within the periodic group. For example, in the NATURAL DDM named CUSTOMERS, the SIGNATURE-LIST data field is a periodic group for data fields LIMIT and SIGNATURE. PROC ACCESS creates a SAS variable named SL_OCCUR for the occurrence numbers. For more information and examples, see “Using Periodic Group Fields in Selection Criteria” on page 129.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ACCESS® Interface to ADABAS Software: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999.

SAS/ACCESS® Interface to ADABAS Software: Reference, Version 8

ISBN 1–58025–546–9

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999