CHAPTER 6

ACCESS Procedure Reference

Introduction

The ACCESS procedure enables you to create and edit the descriptor files used by the SAS/ACCESS interface view engine to IMS-DL/I (hereafter referred to as the IMS-DL/I engine). The ACCESS procedure can be used in the PROGRAM EDITOR and in batch, interactive line, and noninteractive modes.

This chapter provides complete reference information for the ACCESS procedure. The PROC ACCESS statement is presented first, followed by the statement options and procedure statements. For examples of how to use the statement options, refer to Chapter 3, "Defining SAS/ACCESS Descriptor Files," on page 39. "Performance and Efficient View Descriptors" on page 122 presents several efficiency considerations for using the SAS/ACCESS interface to IMS-DL/I.

Refer to the *SAS Language Reference: Dictionary* and the *SAS Companion for the OS/390 Environment* for information about SAS data sets, data libraries, and their naming conventions, or for help with the terminology used in this procedure description.

Remember that help is available from within the ACCESS procedure if you type **HELP** on any command line.

ACCESS Procedure Syntax

PROC ACCESS <*options*>;

Creating or Updating Statement

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

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

Database-Definition Statements

DATABASE=*database-name* DBTYPE=*database-type*;

RECORD=*record-name* SEGMENT=*segment-name*
SEGLNG=*segment-length*;

GROUP=*group-name* LEVEL=*level-number*
KEY=Y | N | U OCCURS=*number-of-repeats*
SEARCH=*search-name*;

ITEM=*item-name* LEVEL=*level-number*
DBFORMAT=*database-format*
FORMAT=*SAS-format* SEARCH=*search-name*
KEY=Y | N | U OCCURS=*number-of-repeats*
DBCONTENT=*database-content*;

DELETE *item-name* | *index-number*;

INSERT*item-name* | *index-number*;

REPLACE *item-name* | *index-number*;

Editing Statements

AN=Y | N;

UN=Y | N;

DROP *item-name* | *index-number...* ;

FORMAT *item-name* | *index-number* <=> *format...* ;

LIST ALL | VIEW | *index-number* | *item-name* <*blanks* | DB | DESC>;

QUIT;

RENAME *item-name* | *index-number* <=> *SAS-name...* ;

RESET ALL | *item-name* | *index-number* ... ;

SELECT ALL | *item-name* | *index-number...* ;

SUBSET *selection-criteria*;

RUN;

Description

The ACCESS procedure is used 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 following sections provide more information about the syntax of the PROC ACCESS statement.

PROC ACCESS Statement Options

To create and edit access and view descriptor files, you must issue the PROC ACCESS statement with options and procedure statements. The statement has this format:

PROC ACCESS <*options*>;

required-procedure-statements;

optional-procedure-statements;

This section describes PROC ACCESS options. For information on the procedure statements, see "Procedure Statements" on page 98.

Options

Depending on which options you choose, the ACCESS procedure performs several tasks. To create and edit access and view descriptors, use the following options:

DBMS=*IMS*

specifies the name of the database management system that the access descriptor will access. Specify DBMS=IMS since you are using the SAS/ACCESS interface to IMS-DL/I.

ACCDESC=*libref.access-descriptor*

specifies the name of 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 data set option on the ACCDESC= option to specify any passwords that have been assigned to the access descriptor. The access descriptor that you name must exist.

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

The following options allow you to extract IMS-DL/I data with a view descriptor:

VIEWDESC=<*libref.*>*view-descriptor*

specifies the name of the view descriptor from which to extract the IMS-DL/I data.

OUT=<*libref.*>*member-name*

specifies the SAS data file to which DBMS data are written. OUT= is used only with the VIEWDESC= option.

CAUTION:

Altering an IMS-DL/I database invalidates descriptors. Altering an IMS-DL/I database that has descriptor files defined on it may cause these descriptors to be out-of-date or invalid. For example, if you add an item to a database segment and an existing access descriptor is defined on that database, the access descriptor does not reflect the new item. △

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, as described in “Assigning Passwords” on page 96.

Table 6.1 on page 96 summarizes the levels of protection that SAS System passwords have and their effects on access descriptors and view descriptors.

Table 6.1 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=ims accdesc=mylib.account(alter=rouge);
  create vlib.customer.view;
  select all;
run;
```

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

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

Assigning Passwords

You can assign, change, or clear a password for an access descriptor, a view descriptor, or another SAS file by using the DATASETS procedure’s MODIFY statement. 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 REWARD with the ALTER level of protection to the access descriptor MYLIB.EMPLOYEE:

```
proc datasets library=mylib memtype=access;
   modify employee (alter=reward);
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 MYLIB.EMPLOYEE.

You can assign multiple levels of protection to a descriptor or SAS data file. See “Ensuring Data Security” on page 130 for more information about how to prevent unauthorized access to the data in your IMS-DL/I databases.

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.CUSTACCT:

```
proc datasets library=vlib memtype=view;
   modify custacct (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.CUSTACCT 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.CUSTACCT, 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 custacct (read=mypw/ alter=mydept/);
run;
```

In the following example, PROC DATASETS sets a READ and ALTER password for view descriptor VLIB.CUSTINFO. PROC PRINT tries to use the view descriptor with both an invalid and valid password. PROC ACCESS tries to update the view descriptor with and without a password.

```
/* Assign passwords */
proc datasets library=vlib memtype=view;
  modify custinfo (read=r2d2 alter=c3po);
run;

/* Invalid password given */
proc print data=vlib.custinfo (pw=r2dq);
  where soc_sec_number = '178-42-6534';
  title2 'Data for 178-42-6534';
run;
/* Valid password given */
proc print data=vlib.custinfo (pw=r2d2);
  where soc_sec_number = '178-42-6534';
  title2 'Data for 178-42-6534';
run;

/* Missing password */
proc access dbms=ims;
   update vlib.custinfo.view;
     drop country;
     list all;
run;
```

```
/* Valid password given */
proc access dbms=ims;
   update vlib.custinfo.view (alter=c3po);
     drop country;
     list all;
run;
```

Refer to *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 95 and certain procedure statements. The options and statements you choose are defined by your task.

- To create an access descriptor:

  PROC ACCESS DBMS=*IMS*;

  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=*IMS*;

  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=*IMS* ACCDESC=*libref.access-descriptor*;

  CREATE *libref.member-name*.VIEW;
  SELECT *item-list;*
  optional editing statements;

  RUN;

- To update an access descriptor:

  PROC ACCESS DBMS=*IMS*;

  UPDATE *libref.member-name*.ACCESS;
  procedure statements;

RUN;

- To update a view descriptor:

  PROC ACCESS DBMS=*IMS*;
  UPDATE *libref.member-name*.VIEW;
  procedure statements;

  RUN;

Database-Description Statements

The following statements define the IMS-DL/I database in an access descriptor.

DATABASE=*database-name* DBTYPE=*database-type*;

RECORD=*record-name* SEGMENT=*segment-name*
SEGLNG=*segment-length*;

GROUP=*group-name* LEVEL=*level-number*
KEY=Y | N | U OCCURS=*number-of-repeats*
SEARCH=*search-name*;

ITEM=*item-name* LEVEL=*level-number*
DBFORMAT=*database-format*
FORMAT=*SAS-format* SEARCH=*search-name*
KEY=Y | N | U OCCURS=*number-of-repeats*
DBCONTENT=*database-content;*

DELETE *item-name* | *index-number;*

INSERT *item-name* | *index-number;*

REPLACE *item-name* | *index-number;*

The DATABASE=, RECORD=, and ITEM= statements are required to create an access descriptor with the CREATE statement; the GROUP= statement is optional. The INSERT, DELETE, and REPLACE statements are used with the UPDATE statement to change an existing access descriptor. At least one of the GROUP=, RECORD=, or ITEM= statements must be used with the INSERT, DELETE, and REPLACE statements to change an access descriptor. The DATABASE= statement cannot be used in an UPDATE statement.

Whether you are creating or changing an access descriptor, the RECORD=, ITEM=, and GROUP= statements must be used in the same order as they appear in the database.

Because IMS-DL/I does not have a dictionary or store descriptive information about the database, you need to provide the DBD information. To provide this information, you need to have a COBOL copybook or layout of the database.

For logical databases, the access descriptor definitions are mapped to the logical DBD and not to one or more physical DBDs. This enables the IMS-DL/I engine to build correct calls and for the SSAs (segmented search arguments) to navigate the logical structure of the database.

Note: See "Tools for Creating Access Descriptors" on page 121 for tools that SAS Institute supplies to automate the database definition process. △

Editing Statements

SAS/ACCESS *editing statements* enable you to drop or rename items, list items, reset names, and so on in a descriptor. All of the statements can be used when you are creating a descriptor. The ASSIGN=, SELECT, RESET, and UNIQUE= statements cannot be used when you are changing a descriptor.

When creating or changing an access descriptor, place editing statements after the last database definition statement. All editing statements are optional.

The following list shows the basic syntax of each editing statement:

ASSIGN=Y| N;

UNIQUE=Y| N;

DROP *item-name* | *index-number. . .* ;

FORMAT *item-name* | *index-number* <=> *format. . .* ;

LIST ALL | VIEW | *index-number* | *item-name* <*blanks* | DB | DESC>;

QUIT | EXIT;

RENAME *item-name* | *index-number* <=>
SAS-name. . . ;

RESET ALL | *item-name* | *index-number. . .* ;

SELECT ALL | *item-name* | *index-number. . .* ;

SUBSET *selection-criteria*;

These statements are described in detail in the following sections.

Dictionary

ASSIGN= | AN=

Generates SAS names and formats that are based on item names and DB Formats

Optional statement

Applies to: access descriptor

Syntax

ASSIGN| AN=Y | N;

Details

The ASSIGN= statement causes view descriptors to inherit the SAS variable names and formats of the parent access descriptor at the time that the descriptor is created. That is, if ASSIGN=Y, the variable names generated for the access descriptor will be used in all derived view descriptors, regardless of the naming conventions used.

If ASSIGN=N, which is the default value, you specify the SAS variable names and formats when you create a view descriptor from this access descriptor. The naming conventions used by the view descriptors are determined by examining the VALIDVARNAME SAS System option. The VALIDVARNAME SAS System option lets users specify what naming conventions will be allowed in a SAS session, either Version 6, Version 7, or a Version 7 option, and enforces them by converting variable names that do not conform to the necessary format. For more information on the VALIDVARNAME SAS System option, see "Overview of Using the Interface" on page 4 and *SAS Language Reference: Dictionary*.

If you enter a value of Y for this statement, you cannot specify the RENAME, FORMAT, and UN= statements when creating view descriptors that are based on this access descriptor.

When a new CREATE statement is entered, the ASSIGN= statement is reset to the default value, N.

CREATE (Access Descriptor)

Creates an access descriptor

Required statement

Applies to: access descriptor

Syntax

CREATE *libref.member.*ACCESS;

Details

The CREATE statement specifies a one- or two-level name for the access descriptor you want to create. The suffix specifies the member type ACCESS. You can use the CREATE statement in one procedure execution as many times as necessary.

To create an access descriptor, the CREATE statement must follow the PROC ACCESS statement. It is specified before any of the database description or editing statements, which are described later in this chapter.

When you submit a CREATE statement for processing, the statement is checked for errors and, if none are found, the access descriptor specified in the previous CREATE statement (if there is one) is saved. If errors are found, error messages are written to the SAS log and processing is terminated. After you correct the error, resubmit the statements or batch job for processing.

CREATE (View Descriptor)

Creates a view descriptor

Required statement

Applies to: view descriptor

Syntax

CREATE *libref.member.*VIEW PSBNAME=*psb-name* <PCBINDEX=*pcb-index*>
<GSAM>
;

Details

The CREATE statement specifies a one- or two-level name for the view descriptor you want to create. The suffix specifies the member type VIEW. This statement is required to create and save a view descriptor.

To create a view descriptor, add the CREATE statement after the procedure statements that create the access descriptor on which this view descriptor is based. If you are creating a view based on an existing access descriptor, specify the access descriptor's name in the ACCDESC= option in the PROC ACCESS statement.

Place any editing statement and view-descriptor-specific statements, such as the SELECT and SUBSET statements, after the view descriptor's CREATE statement. You can submit more than one CREATE statement in one execution of the PROC ACCESS statement. As with other SAS procedures, end the ACCESS procedure with a RUN statement.

When you submit a CREATE statement for processing, the statement is checked for errors and, if none are found, the view descriptor specified in the previous CREATE statement (if there is one) is saved. If errors are found, error messages are written to the SAS log and processing is terminated. After you correct the error, resubmit the statements or batch job for processing.

Arguments

The following list explains the arguments that can appear in a CREATE statement for a view descriptor:

PSBNAME= PSB=
: specifies the name of the PSB that references the IMS-DL/I database on which this view descriptor is based. This is a required argument.

PCBINDEX= PCB=
: specifies the PCB in the PSB that references the database. This argument is optional; you need to specify a PCB index only if the PSB references the database

more than once. If you do not specify a PCB index and the PSB references the database more than once, the first PCB in the PSB that references the database is used.

GSAM
: specifies that the database on which this descriptor is based is a GSAM database. Specify this argument only if you have a GSAM database.

DATABASE=

Specifies the DBD name of the IMS-DL/I database on which this access descriptor is based

Required statement

Applies to: access descriptor

Syntax

DATABASE=*database-name* DBTYPE=*database-type*;

Details

The DATABASE= statement specifies the DBD name of the IMS-DL/I database on which this access descriptor is based. DBD= is an alias for the DATABASE= statement. If you are creating an access descriptor, the DATABASE= statement must be the first statement after the CREATE= statement.

For logical databases, the access descriptor definitions are mapped to the logical DBD (database description) and not to one or more physical DBDs. This enables the IMS-DL/I engine to build correct database calls and for the SSAs (segmented search arguments) to navigate the logical structure of the database.

Note: See "Tools for Creating Access Descriptors" on page 121 for tools that SAS Institute supplies to automate the database definition process. △

Arguments

The following list explains the arguments that can appear in a DATABASE= statement for an access descriptor:

DBTYPE= DBT=
: specifies the type of database and is required with the DATABASE= statement. Valid database types are HDAM, HIDAM, HSAM, HISAM, GSAM, SHSAM, and SHISAM. See "IMS-DL/I Database Types" on page 21 for a description of each database type. You can use DBT= as an alias for DBTYPE=.

  DBTYPE= tells the IMS-DL/I engine how to handle WHERE clauses that generate SSAs for database calls. If you omit DBTYPE= from your DATABASE= statement, you receive the following error:

```
ERROR 22-322: Expecting one of the following:
DBTYPE = NAME. The statement is being ignored.

ERROR: Must enter database name first.
```

An example of the DATABASE= statement is

```
database=acctdbd dbtype=hisam;
```

DELETE

Removes records, groups, or items from an existing access descriptor

Optional statement

Applies to: access descriptor

Interacts with: UPDATE statement

Syntax

DELETE | DEL *numeric-list*;

DELETE | DEL *item-name* <... *item-name-n*>;

Details

The DELETE statement deletes the specified record, group, or item from an access descriptor. You can specify as many records, groups, or items as you want in one DELETE statement. When you delete a group or record, all of the items in that group or record are deleted as well.

Note that if the first record of a descriptor is deleted, then the first item in the descriptor must still be a RECORD.

Arguments

The following arguments can appear in a DELETE statement. You can mix item names and quoted strings in the same DELETE statement, but you cannot mix index numbers and names. Referencing a list of index numbers is an efficient way to delete items like OCCURS clauses, which by definition are not unique.

numeric-list
is a list of index numbers, optionally separated by logical operators, that represent the item's place in the access descriptor. You can obtain the index number of an item using the LIST statement described later in this section.

item-name
is the name of the IMS-DL/I group, record, or item to be deleted. This field can also contain a quoted string.

Examples

The following are examples of DELETE statements:

```
DELETE 15 2 8 TO 12;       /* deletes a numeric list */
DELETE 1 TO 23 BY 2;       /* deletes a numeric list */
DELETE CITY STATE ZIP;     /* deletes by name        */
DELETE CITY 'FIRST-ORDER-DATE';
                 /* deletes a name and quoted string */
```

DROP

Drops the specified item so that it is no longer available for selection

Optional statement

Applies to: access descriptor or view descriptor

Syntax

DROP *numeric-list*;

DROP *item-name* <... *item-name-n*>;

Details

The DROP statement drops the specified item so that the item is no longer available for selection. When used in an access descriptor, it prevents the specified item from being available to a view descriptor. The DROP statement is used with the UPDATE statement in a view descriptor.

You can specify as many items to be dropped as necessary by using one DROP statement. You can identify items by their index number or by their name or a quoted string, but you cannot mix index numbers and names. If you drop a record or group, all the items in that record or group are dropped.

Arguments

The following arguments can appear in the DROP statement:

numeric-list
: is a list of index numbers, optionally separated by logical operators, that represent the item's place in the descriptor. You can get the index number of an item by using the LIST statement described later in this section.

item-name
: is the name of the IMS-DL/I item to be dropped or a quoted string.

FORMAT

Assigns a SAS format to an IMS-DL/I item

Optional statement

Applies to: access descriptor or view descriptor

Syntax

FORMAT *item-name* | *index-number* <=> *format* <... *item-name-n* | *index-number-n* <=> *format-n*>;

Details

The FORMAT statement assigns a SAS format to an IMS-DL/I item. You can assign formats to as many items as necessary using one FORMAT statement. Note that the equal sign (=) between arguments is optional. You cannot use the FORMAT statement for a record or group.

Arguments

The following list explains the arguments that can appear in the FORMAT statement:

item-name
: is the name of the IMS-DL/I item for which you want to assign or change the SAS format.

index-number
: is the index number of the IMS-DL/I item for which you want to assign or change the SAS format. The index number represents the item's place in the access descriptor. You can get the index number of an item using the LIST statement described later in this section.

format
: is the SAS format that you want to assign to the specified IMS-DL/I item.

GROUP=

Defines the groups within the record

Optional statement

Applies to: access descriptor

Syntax

GROUP= *group-name* LEVEL=*level-number* <KEY=Y|N|U>
 <OCCURS=*number-of-repeats*> <SEARCH=*search-name*>;

Details

The GROUP= statement defines the groups within the record. This statement is optional.

See "Handling GROUP Keys in Descriptor Files" on page 135 for information about how to reference GROUP keys in a view descriptor's WHERE statement.

Arguments

In the GROUP= statement, you must enter the group name and level number. The other arguments are used to further define the group and are not required. The following list explains each argument that can appear in a GROUP= statement:

GROUP=
GR=
: is the name that you want to assign to the group item in an IMS-DL/I database. This name can be a maximum of 32 characters. If any special characters or blanks

are included in the name, enclose the entire name in quotation marks. This is a required argument.

LEVEL=
LV=
: is the two-character numeric level of the IMS-DL/I item. This level number is similar to the COBOL level number. Groups have levels greater than 01, and their level numbers are less than the level numbers of the items within the group. This is a required argument.

KEY=
K=
: indicates with an Y, N, or a U whether this item is defined in the DBD as a sequence or key field and whether the key sequence field is unique. The default setting, N, indicates the field is not a key sequence field. You must assign one key sequence field per segment if you plan to use the view descriptors that are created from this access descriptor to update the IMS-DL/I database. Keys are recommended, but not required, for all segments except the lowest hierarchical level if the view descriptors will be used only for data retrieval. When KEY=U, retrieval calls to IMS will be reduced because the IMS engine will know that there is only one segment in the database for this key.

OCCURS=
O=
: indicates the number of times a repeating group occurs. This is an optional argument.

SEARCH=
SE=
: is the search field name defined for the group item in the DBMS DBD. If you want the IMS-DL/I engine to create SSAs directly from a WHERE statement or command, you must enter the search field names. Otherwise, the WHERE statement is passed to the SAS System and all of the segments in the database that are referenced in the view descriptor are read. SEARCH= is an optional argument, but it is recommended where applicable.

Note: See "Handling GROUP Keys in Descriptor Files" on page 135 for important information about searching at the GROUP level. Also see "Performance and Efficient View Descriptors" on page 122 for more information about SSAs and WHERE statements. for important information about searching at the GROUP level. △

INSERT

Adds new records, groups, or items to an existing access descriptor.

Optional statement

Applies to: access descriptor

Interacts with: UPDATE statement

Syntax

INSERT | INS *index-number*;

INSERT | INS *item-name* <... *item-name-n*>;

Details

The INSERT statement is a positioning statement; it inserts the RECORD=, GROUP=, or ITEM= statements following it after the item it references. The syntax and use of the RECORD=, GROUP=, and ITEM= statements are the same in update mode as they are in create mode.

Although the INSERT statement can reference only one item, more than one RECORD=, GROUP=, or ITEM= statement can follow an INSERT statement. The INSERT statement retains control until it encounters an editing, LIST, DELETE, or REPLACE statement, or the ACCESS procedure ends through a QUIT, RUN, or other procedure statement. Multiple INSERT statements can be used in one UPDATE statement. When more than one INSERT statement references the same item, the most recent update displays as first.

Arguments

The following arguments can appear in a INSERT statement:

item-number
: is an index number that represents the item's place in the access descriptor. You can get the index number of an item by using the LIST statement described later in this section.

item-name
: is the name of the IMS-DL/I group, record, or item after which subsequent groups, records, or items will be inserted. This field can also contain a quoted string.

Example

The following is an example of an INSERT statement. A new record and item are inserted at the beginning of access descriptor ADLIB.CUSTINS. "INSERT 0" inserts items at the beginning of the descriptor. The first item in an access descriptor must always be a record. Also in the example, note that the first LIST statement prints a pre-update listing of the database as defined by the access descriptor, while the second prints a post-update listing.

```
proc access dbms=ims;
   update adlib.custins.access;
      list all db;
      insert 0;
        record=newfrec sg=newrecsg sl=400;
        item=newfitem lv=3 dbf=$12. se=custfsti;
      list all db;
run;
```

ITEM=

Defines the fields within the record

Required statement

Applies to: access descriptor

Syntax

ITEM= *item-name* LEVEL=*level-number*
DBFORMAT=*database-format* <SASNAME=*SAS-name*>
<FORMAT=*SAS-format* > <SEARCH=*search-name*>
<KEY=Y|N|U> <OCCURS=*number-of-repeats*>
<DBCONTENT=*database-content* >;

Details

The ITEM= statement defines the fields within the record.

Arguments

In the ITEM= statement, you must enter the item name, level number, and the DBFORMAT= argument. The other arguments are used to further define the item and are not required. The following list explains each argument that can appear in the ITEM= statement:

ITEM=
IT=

is the name that you want to assign to the field in an IMS-DL/I database segment and which the SAS/ACCESS software will use to generate a SAS variable name. This name can be a maximum of 32 characters. If any special characters or blanks are included in the name, enclose the entire name in single quotes. This is a required argument.

The generated SAS variable name will use the following naming conventions specified by the VALIDVARNAME SAS System option:

- □ If VALIDVARNAME=V7, then the variable names will consist of mixed-case, alphanumeric characters and underscores for non-conforming characters, with a letter or underscore in the first position.
- □ If VALIDVARNAME=UPCASE, then the variable names will use Version 7 conventions with uppercase letters.
- □ If VALIDVARNAME= ANY, then the variable names can consist of any characters as long as they are enclosed in quotes and are followed by an n-literal, for example, 'string'n.
- □ If VALIDVARNAME=V6, then the SASNAME= argument assigns your SAS variables names, if used, or the ITEM name is truncated to 8 characters.

If you specified the AN= statement with a value of Y, you will not be able to change the SAS variable names when you create a view descriptor from this access descriptor after the access descriptor statements have been entered.

If you specified the UN= statement with a value of Y, the variable names will be unique. Any duplicate names will be resolved as follows: the name will be truncated to the legal length and a number appended to the end to identify it as unique. For example, two instances of CUSTOMER_ADDRESS would be changed to CUSTOMER_ADDRESS and CUSTOMER_ADDRESS0.

Sites commonly refer to undesired portions of the data buffer by using the FILLER notation in the ITEM= statement and by defining the DBC (DB Content) as $CHAR. See "Using Filler Notation in ITEM=" on page 137 for information.

LEVEL=
LV=
is the two-character numeric level of the IMS-DL/I field. This level number is similar to the COBOL level number. To indicate that a field is in a group, it should have a level number greater than the group's level number. This is a required argument.

DBFORMAT=
DBF=
is used to specify how the IMS-DL/I field is stored in the database. See "Using IMS-DL/I Data Types in SAS/ACCESS Descriptors" on page 22 for a table of recommended DB formats to use for COBOL and PL/I data types. This table also shows the SAS variable formats that the SAS/ACCESS interface to IMS-DL/I generates for the DB Formats.

You must specify one of the following valid SAS informats in this argument. For character data, the valid SAS informats are

$*w*.	$HEX*w*.
$CHAR*w*.	$PHEX*w*.
$CHARZB*w*.	

For numeric data, the valid SAS informats are

w.d	ZDB*w.d*	RB*w.d*
F*w.d*	IB*w.d*	PD*w.d*
BZ*w.d*	PIB*w.d*	PK*w.d*
ZD*w.d*	HEX*w*.	

SASNAME=
SN=
this argument is supported for Version 6 compatibility only. It assigns a SAS variable name to this IMS-DL/I field. When VALIDVARNAME=V6, the name assigned to this argument is also used as input to the subsetting WHERE statement.

FORMAT=
FMT=
assigns a SAS format to the SAS variable. This is an optional argument.

If you specified the AN= statement with a value of Y, the SAS System assigns default formats (based on the field's database format) to the variables when the access descriptor is created. If you want, you can enter formats using the FORMAT= argument in the ITEM= statement at that time. However, you will not be able to change these formats when you create a view descriptor from this access descriptor once the access descriptor is created.

SEARCH=
SE=
is the search field name defined for the field in the DBMS DBD. If you want the IMS-DL/I engine to create SSAs directly from a WHERE statement or command that references the named item, then you must assign search field names. Otherwise, the WHERE statement is passed to the SAS System, and all occurrences of the segments referenced in the view descriptor in the database are read and passed to the SAS System for further processing. See "Performance and

Efficient View Descriptors" on page 122 for more information about SSAs and WHERE statements. This is an optional argument.

KEY=
K=

indicates with an Y, N, or a U whether this field is defined in the DBD as a sequence or key field and whether the key sequence field is unique. The default setting, N, indicates the field is not a key sequence field. You must assign one key sequence field per segment if you plan to use the view descriptors created from this access descriptor to update the IMS-DL/I database. Keys are recommended, but not required, for all segments except the lowest hierarchical level if the view descriptors will be used only for data retrieval. When KEY=U, retrieval calls to IMS will be reduced because the IMS engine will know that there is only one segment in the database for this key.

OCCURS=
O=

indicates the number of times a repeating field occurs. This is an optional argument.

DBCONTENT=
DBC=

indicates that the values for this field need special handling by the IMS-DL/I engine. This is an optional argument. You can use this argument to specify a SAS format that indicates the way date values are represented internally in the IMS-DL/I database, or to indicate how a field is initialized or stored in the database. This is not the same as the value you entered in the DBFORMAT= argument.

For example, you would use the DBFORMAT= argument to specify that a date is stored as a packed decimal. You would then use the DBCONTENT= argument to indicate where the month, day, and year are stored in that packed decimal. Valid parameters for date values are

YYMMDD6.	MMDDYY8.	JULIAN5.
YYMMDD8.	DDMMYY6.	JULIAN7.
MMDDYY6.	DDMMYY8.	

Valid parameters for special formats values that indicate how a field is initialized are

B	when values are blanks for zero
H	for high values
L	for low values.

These special formats affect how the SAS System displays and updates the fields in the database. Use special format B to indicate to the IMS-DL/I engine that a numeric variable has blanks when its value is zero. Use the special codes H and L to indicate that a variable is initialized to high or low values, respectively.

For example, if you specify L for a variable, the SAS System displays a missing value when it finds low values (hexadecimal zeroes) in the variable. If you update that variable with a missing value, the IMS-DL/I engine writes low values to the variable in the database. If you specify H for a variable, the SAS System displays a missing value when it finds high values (hexadecimal Fs) in the variable. If you update that variable with a missing value, the IMS-DL/I engine writes high values to the variable in the database.

You can also use the special formats values when a date is initialized in a special way. For example, if you had a date initialized to low values, you would enter, enclosed in single quotes, the date format followed by a slash (/) and an initialization code. For example, for an eight-digit date in the MMDDYY8. form initialized to low values, you would enter the following value for the DBCONTENT argument:

```
'MMDDYY8./L'
```

Do not specify a DBCONTENT for records and groups.

LIST

Lists all or selected items in the descriptor and information about the items

Optional statement

Applies to: access descriptor or view descriptor

Syntax

LIST <ALL | VIEW | *index-number* | *item-name*> <*blanks* | DB | DESC>;

Details

The LIST statement lists all or selected items in the descriptor and information about the items.

Arguments

The LIST statement consists of two sets of arguments. Select one argument from the first set to select the items to be displayed, and select one argument from the second set to specify the type of information to be displayed about the selected items.

The first set includes the following arguments:

ALL
: lists all the items in the access descriptor that are available for selection. If an item is dropped, **`NON-DISPLAY`** is displayed next to the item's description when listing an access descriptor. When listing a view descriptor, dropped items are not displayed.

VIEW
: lists all the items in the access descriptor that are selected for the view descriptor.

index-number
: specifies the index number that corresponds to the IMS-DL/I item for which you want to display the current status. The index number represents the item's place in the descriptor.

item-name
: specifies the name of an IMS-DL/I item for which you want to display the current status.

The second set includes the following arguments:

blanks	lists the SAS information, including the DB Format and SAS format information, for the specified items. To use this argument, include only the ALL, VIEW, *item-name*, or *index-number* argument from the first set to specify the items.
DB	lists the database information, including the DB Content, segment name, search field, segment length, key field, and occur field information, for the specified items. Use the ALL, VIEW, *item-name* or *index-number* argument before this argument to specify which items to list.
DESC	lists both SAS and database information for the specified items. Use the ALL, VIEW, *item-name* or *index-number* argument before this argument to specify which items to list.

Note: The LIST statement output is written to the SAS log. △

QUIT

Terminates the procedure without any further descriptor creation

Optional statement

Syntax

QUIT | EXIT;

Details

The QUIT statement terminates the procedure without any further descriptor creation. EXIT is an alias for the QUIT statement.

RECORD=

Defines an IMS-DL/I segment

Required statement

Applies to: access descriptor

Syntax

RECORD=*record-name* SEGMENT=*segment-name* SEGLNG=*segment-length*;

Details

The RECORD= statement defines an IMS-DL/I segment. A value of 01 is automatically assigned as the level number of a record, so the RECORD= statement does not include a level number argument. You should begin your database definition with a RECORD= statement immediately after the DATABASE= statement.

Arguments

The following list explains each of the arguments that can appear in a RECORD= statement.

RECORD= RE=
: specifies an arbitrary name for the segment. A record name can be a maximum of 32 characters. If special characters or blanks are included in the name, enclose the entire name in single quotes. This is a required argument.

SEGMENT= SG=
: specifies the name of the segment as defined in the DBD. A segment name can be a maximum of eight characters. If your database is a GSAM database, enter **GSAM** as the segment name. This is a required argument.

SEGLNG= SL=
: specifies the segment length as defined in the DBD. This is a required argument. See “Handling Segments of Varying Length” on page 135 for more information.

RENAME

Enters or modifies the SAS name for an item

Optional statement

Applies to: access descriptor or view descriptor

Syntax

RENAME *item-name* | *index-number* <=> *SAS-name*
    <... *item-name-n* | *index-number-n* <=> *SAS-name-n*>;

Details

The RENAME statement enters or modifies the SAS variable name for an item. If you are creating a view descriptor from an existing access descriptor with an ASSIGN value of Y, you cannot use the RENAME= statement. You can rename as many items as necessary using one RENAME= statement.

Note: If the VALIDVARNAME SAS System option is set to V6, this statement affects the SAS name parameter; if VALIDVARNAME=V7, it affects the item name. △

Arguments

The following list explains the arguments that appear in the RENAME= statement:

item-name
: is the name of the IMS-DL/I item that you want to rename.

index-number
: is the index number of the IMS-DL/I item that you want to rename. The index number represents the item's place in the descriptor. You can get the index number of an item using the LIST statement described earlier in this section.

SAS-name
: is the new SAS variable name that you want to assign to the specified item.

REPLACE

Modifies record, group, and item definitions in an existing access descriptor

Optional statement

Applies to: access descriptor

Interacts with: UPDATE statement

Syntax

REPLACE | REPL *index-number*

<GROUP= *new-group-name* ITEM=*new-item-name*
LEVEL=*level-number*
DBFORMAT=*database-format*
FORMAT=*SAS-format* SEARCH=*search-name*
KEY=Y|N|U DBCONTENT=*database-content*>;
|
<RECORD=*new-record-name*
SEGMENT= *segment-name*
SEGLNG= *segment-length*>;

REPLACE|REPL *item-name*

<LEVEL=*level-number* DBFORMAT=*database-format*
FORMAT=*SAS-format* SEARCH=*search-name*
KEY=Y|N|U DBCONTENT=*database-content*>;
|
<SEGMENT=*segment-name* SEGLNG=*segment-length*>;

Details

The REPLACE statement replaces or modifies existing records, groups, and items in existing access descriptors. Any item that can be entered on RECORD, GROUP=, and ITEM= statements can be modified, except the OCCURS option.

Unlike the INSERT and DELETE statements, each data item to be modified needs a separate REPLACE statement, although any number of REPLACE statements can occur in any order with INSERT and DELETE statements within an UPDATE statement

Arguments

The only required item on the REPLACE statement is the index number, name, or quoted string used to identify it. However, the optional arguments are recommended for data definition. Except for the following optional arguments, the arguments follow the same editing rules as they would in create mode or in an update insert situation.

- KEY=N will remove an item as a designated key field.
- Specifying blanks on a SEARCH or DBCONTENT parameter removes their value, effectively dropping the parameters.
- The FORMAT parameter currently cannot be reset to its default value.

Examples

The following are examples of replacement statements:

```
replace shipped dbc=mmddyy6.; /* modifies dbcontent */

replace 5 se=' '    /* drops search field parameter */

replace 'old-record-name' record='new-record-name;
   sg='new-ims-segname';      /* replaces record    */

replace 2 item='cust-item';  /* renames item        */
```

RESET

Resets specified or all items to their default settings

Optional statement

Applies to: access descriptor or view descriptor

Syntax

RESET ALL | *item-name* | *index-number*
    <... *item-name-n* | *index-number-n*>;

Details

The RESET statement resets specified or all items to their default settings. You can reset as many items as necessary using one RESET statement or the ALL option to reset all the items. The RESET statement has different effects on access and view descriptors.

Access descriptors In access descriptors, the default setting for a SAS variable name is a blank unless you included the AN= statement. If you used the AN= statement, the names are reset to those generated. The default setting for SAS formats in access descriptors is determined by the DB Formats of the items. Any dropped items will be included again.

View descriptors In view descriptors, the RESET statement deselects items and resets the SAS name and format values to those defined in the access descriptor on which the view descriptor is based. The SAS names and formats are unaffected by the RESET statement if you specified the AN= statement with a value of Y when you created the access descriptor on which this view descriptor is based.

Arguments

The following list explains the arguments that appear in the RESET statement:

ALL
: resets all the items defined in the access descriptor to their default setting. For a view descriptor, the ALL option resets only the items that are selected.

item-name
: specifies the name of the item that you want to reset. If you specify a record or group name, all the items in that record or group are reset.

index-number
: specifies the index number of the item that you want to reset. The index number represents the item’s place in the access descriptor. You can get the index number of an item using the LIST statement described earlier in this section. If you specify a record or group index number, all the items in that record or group are reset.

SELECT

Selects the items in the access descriptor that are to be included in the view descriptor

Optional statement

Applies to: view descriptor

Syntax

SELECT ALL | *item-name* | *index-number* <... *item-name-n*|*index-number-n*>;

Details

The SELECT statement selects the items in the access descriptor that are to be included in the view descriptor. Use the SELECT statement only when you are defining view descriptors. You can select as many items as necessary using one SELECT statement.

Arguments

The following list explains the arguments that appear in the SELECT statement:

ALL
: includes in the view descriptor all of the items that are defined in the access descriptor that were not dropped.

CAUTION:
If the access descriptor contains segments representing more than one path, using ALL will create an invalid view descriptor. △

item-name
specifies the name of the item you want to select to be included in the view descriptor. If you specify a record or group name, all the items in that record or group are selected.

index-number
specifies the index number of the item you want to select. The index number represents the item's place in the access descriptor. You can get the index number of an item using the LIST statement described earlier in this section. If you specify a record or group index number, all the items in that record or group are selected.

SUBSET

Adds or modifies selection criteria defined for a view descriptor

Optional statement

Applies to: view descriptor

Syntax

SUBSET <*selection-criteria*>;

Details

The SUBSET statement specifies the selection criteria for the view descriptor. If you do not use the SUBSET statement, the view will include all occurrences of the segments included in the view descriptor.

Arguments

The *selection-criteria* argument can be new or modified selection criteria that you want to define for the view descriptor. Only a WHERE statement can be used with the SUBSET statement.

Use SAS variable names in the SAS WHERE statement to specify selection criteria. Any variable specified in the WHERE statement must also be selected in the view descriptor. If your statement includes a date or time representation, use the SAS date or time constant representation, such as '01JAN91'D.

To improve performance, use WHERE statements from which the IMS-DL/I engine can generate SSAs. For more information about creating efficient view descriptors, see "Performance and Efficient View Descriptors" on page 122. For more information about the WHERE statement and the expressions it allows, see *SAS Language Reference: Dictionary*.

You can delete the current selection criteria by issuing the SUBSET statement without an argument.

UNIQUE | UN=

Generates unique SAS names based on item names

Optional statement

Applies to: view descriptor

Syntax

UNIQUE |UN = Y | N;

Details

The UNIQUE= statement specifies whether unique SAS variable names should be generated for items. The UNIQUE= statement can be used only when creating a view descriptor.

The default value, N, allows you to enter duplicate SAS variable names. You must resolve these duplicate names before you create view descriptors based on the access descriptor.

If you specify a value of Y and duplicate SAS variable names exist, numbers are appended to any SAS names that are duplicated as the result of truncation. For example, if you enter a value of Y for the UNIQUE= statement, two instances of the item ADDRESS would be changed to ADDRESS and ADDRESS0.

Note: If you specified a value of Y for the ASSIGN= statement when you created the access descriptor on which this view descriptor is based, you cannot specify a UNIQUE= statement. △

UPDATE

Updates a SAS/ACCESS descriptor file.

Optional statement

Applies to: access descriptor or view descriptor

Syntax

UPDATE *libref.member.*ACCESS | VIEW;

Details

The UPDATE statement identifies an existing access descriptor or view descriptor that you want to change. The descriptor can exist in 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.

To update a descriptor, use its three-level name. The first level identifies the libref of the data library where you stored the descriptor. The second level is the descriptor's member name. The third level is the type of SAS file: ACCESS or VIEW. For a view descriptor, you can optionally specify the PSBNAME and PCBINDEX arguments.

You can use the UPDATE statement as many times as necessary in one procedure. Use these guidelines to write the UPDATE statement:

- Like the CREATE statement, the UPDATE statement should immediately follow PROC ACCESS and precede any database definition and editing statements. Also, all database definition statements should precede any editing statements.
- Within the database definition group, the DELETE, INSERT, and REPLACE statements can be specified in any order and can occur multiple times with an UPDATE sequence. The order has no effect on processing.
- When using index numbers, the numbers specified with the UPDATE statement refer to the original pre-update order. Index numbers used with editing statements always apply to the post-update, "ready to rewrite" order. See "Editing Statements" on page 100 for a list of editing statements.
- Use the LIST statement after the UPDATE statement and avoid using intermediate LIST statements, particularly in batch mode. The LIST statement forces a reorganization of the in-memory layout of the access or view descriptor. Intermediate list statements change the index numbering at each invocation and can cause an error.
- Do not attempt to 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 examples edit the access descriptor IMSLIB.CUSTS. Despite the order of the INSERT, DELETE, and REPLACE statements in the update sequence, the examples produce identical results.

```
/* ----example 1------ */
proc access dbms=ims;
  update imslib.custs.access;
   insert address;
      item=address2     lv=3 dbf=$12    se=custadd2;
   delete contact;
   repl 23 se=custphon;
   ins 23;
      item=newitem      lv=3 dbf=$30    se=custlsti;
run;

/* ---example 2--- */
proc access dbms=ims;
  update imslib.custs.access;
   delete contact;
   repl 23 se=custphon;
   ins 23;
      item=newitem      lv=3 dbf=$30    se=custlsti;
   insert address;
      item=address2     lv=3 dbf=$12    se=custadd2;
run;
```

The following example shows how index numbers are interpreted by different parts of an UPDATE statement. In the example, the DELETE statement processes the third item in the original descriptor. The DROP statement, however, processes the fourth item in the post-update order, which in this case would have been the fifth item in the original sequence.

```
proc access dbms=ims;
  update imslib.custs.access;
    delete 3;  /* pre-update item 3 */
    drop 4;    /* post-update item 4 */
  list all;
run;
```

Pre-update and post-update listings are shown below.

```
/* ---prior to UPDATE --- */
IMS Database: CUSTOMER
Function: Create Descriptors-access: CUSTS1  view:
   L# Item Name                DBFormat   Format
1  01 CUSTOMER                 *RECORD*   *RECORD*
2  02 CUSTOMER-INFO            *GROUP*    *GROUP*
3  03 CUSTOMER-CODE            $8.        $8.
4  03 STATE                    $2.        $2.
5  03 ZIP                      10.0       12.0
6  03 COUNTRY                  $20.       $20.

/* ---after UPDATE --- */
IMS Database: CUSTOMER
Function: Create Descriptors-access: CUSTS1  view:
   L# Item Name                DBFormat   Format
1  01 CUSTOMER                 *RECORD*   *RECORD*
2  02 CUSTOMER-INFO            *GROUP*    *GROUP*
3  03 STATE                    $2.        $2.
4  03 ZIP   *NON-DISPLAY*      10.0       12.0
5  03 COUNTRY                  $20.       $20.
```

Tools for Creating Access Descriptors

The SAS/ACCESS interface to IMS-DL/I is different from other SAS/ACCESS interfaces in that it requires you to define the database in your access descriptor. Other SAS/ACCESS interfaces are able to query a data dictionary or other information repository to acquire detailed information about the database object that is being accessed.

Defining access descriptors for IMS-DL/I databases can be time consuming because the data have to be entered manually. To automate this process, especially in cases where many access descriptors must be defined, there are several tools available for your use.

COB2SAS Tool

The COB2SAS tool uses the COB2SAS utility to process COBOL copybook database definitions and to store them in a permanent SAS data file. This data file is then processed by a DATA step program that is supplied in the installed *prefix.*SAMPLE PDS, called IMSS2A. The IMSS2A program processes the observations in the data file and generates most of the syntax required by the PROC ACCESS procedure statements that create an access descriptor for the database.

The generated statements are written to a host file (physical sequential or PDS member) where they can be edited. The statements written to the host file require some editing because the copybook file does not contain all the information that is necessary to create the access descriptor. You need to add DBD-specific information such as segment lengths, search and sequence field names, DBD name, DBTYPE, and segment names, in order to complete the code. You can then either submit the generated statements with JCL in a batch execution, or submit them from the SAS PROGRAM EDITOR window.

The COB2SAS tool is available from SAS Institute free of charge for download from the World Wide Web, from an FTP site, or in the form of a mailer tape. This tool was originally designed to aid in converting COBOL file copybooks to INPUT statements for SAS DATA steps. For access descriptor creation, it is not necessary to complete all of the steps outlined in the COB2SAS usage instructions. Typically, after the copybook is processed, the results are stored in a temporary SAS file, which is then used to generate the INPUT statement. For IMS-DL/I access descriptor creation, only the steps up to and including creation of the SAS file (dictionary file) are necessary. A modification is made to make the dictionary file permanent, and from there the IMSS2A program is used to complete the process.

Note that only steps R2COB1-R2COB5 are needed to create the dictionary file. Member R2MVS is the file to edit to make the dictionary a permanent file. R2MVS is also the main program that drives all of the other steps. It is well documented, and comments provide information on what each step does.

For more information about using the COB2SAS tool and about the IMSS2A sample program, call SAS Institute's Technical Support Division.

SAS Macro and DATA Step Code

The second tool was donated by a SAS user.* The tool consists of SAS macro and DATA step code that processes the database DBD directly. The benefit of this tool is that the file of generated PROC ACCESS code does not need further editing before being submitted for execution. This tool is available by request from SAS Institute's Technical Support Division.

Performance and Efficient View Descriptors

When you create and use view descriptors, follow these guidelines to minimize the use of IMS-DL/I and OS/390 system resources and to reduce the time IMS-DL/I takes to access data.

General Information

Select only the items your program needs. Selecting unnecessary fields adds extra processing time.

Sorting data can be resource-intensive, even if it is done using the SORT procedure. You should sort data only when sorted data are needed for your program. Note that IMS-DL/I does not support the ORDER BY clause or a BY statement in an application, such as **`PROC PRINT ... BY`** *variable* **`...;`**. If you have an IMS-DL/I database that does not have an index and you want to use a SAS procedure that requires the data to be sorted, you must first extract the data to sort them. If you have an IMS-DL/I database that does have an index and you want to use a BY variable other than an index key, you must also extract the data to sort them before executing the SAS procedure.

Where possible, specify selection criteria that can be converted into SSAs to subset the amount of data IMS-DL/I returns to the SAS System.

* Bruce Babbitt of New England Power Service Company

Extracting Data Using a View

If a view descriptor describes a large IMS-DL/I database and you will use the temporary or permanent view descriptor many times, it may be more efficient to extract the data and place them in a SAS data file. Under the following circumstances you should probably extract data:

- □ If you plan to use the same IMS-DL/I data in several procedures, you may improve performance by extracting the IMS-DL/I data. Placing the IMS-DL/I data into a SAS data file requires disk space to store the data and I/O to write the data. However, SAS data files are organized to provide optimal I/O performance with PROC and DATA steps. Programs using SAS data files often use less CPU time than programs that directly read IMS-DL/I data.
- □ If you plan to read a large amount of data from a large IMS-DL/I database and the database is being shared by several users, your direct reading of the data could adversely affect all users' response time. Extracting data can improve response time.
- □ If you think directly reading this data would present a security risk, you may want to extract the data and not distribute information about either the access descriptor or view descriptor.

Deciding How to Subset Your Data

There are many reasons why you may want to subset or filter the data being returned from a database path defined by a view descriptor. The main benefit is performance. Retrieving a portion of the data in the database path is more efficient than retrieving all of the data in the path. Another reason is to enforce security measures, such as restricting users of view descriptors to certain subsets of data.

Once you determine that your application can benefit from using a subset of data, there are several ways that you can subset data in the SAS System. Use the following guidelines to determine when to use a view descriptor WHERE expression, an application WHERE expression, a DATA step subsetting IF statement, and when to use a combination of the methods.

Note: Regardless of the method that you choose, for performance reasons you should always attempt to choose selection criteria that can be converted by the engine into SSAs. If the engine cannot build SSAs for your data request, then a sequential access method is used to retrieve all path data defined by the view descriptor. △

View Descriptor WHERE Expression

Include a WHERE expression in your view descriptor by using a SUBSET statement when you want to

- □ have selection criteria that you want to always apply, regardless of the application that references the view descriptor.
- □ restrict access to data in a way that the selection criteria cannot be viewed, modified, or deleted.

Selection criteria stored in a view descriptor can be protected with a password as well as operating system security. If an application specifies additional subset criteria, it is combined with the view descriptor selection criteria and treated as an AND search argument.

Application WHERE Expression

Use an application WHERE expression (SAS WHERE statement, clause, or data set option) when the guidelines specified in the previous section do not apply and you

- □ want to use the same view descriptor for various tasks (includes DATA steps, procedures, and SCL), where each requires a different subset of data
- □ need to generate dynamic selection criteria for the data defined by the view descriptor.

For a more detailed description of how the WHERE expressions work, see "WHERE Statement Processing" on page 142.

DATA Step IF Statement

Use a subsetting IF statement in a DATA step execution when you

- □ need to impose selection criteria that would result in a sequential retrieval of the data defined by the view descriptor. This type of criteria does not meet SSA eligibility requirements.

The IMS-DL/I engine generates SSAs only when all of the conditions in a WHERE expression meet eligibility requirements. The DATA step IF statement allows you to perform filtering that does not meet SSA eligibility requirements, while using a view descriptor WHERE expression or application WHERE expression to obtain the performance benefits from SSAs.

Combination of Methods

There are some comparison operators in the SAS System that cannot be incorporated into SSAs for DL/I function calls and that cannot be used with the DATA step IF statement. In these cases, you will have to evaluate the impact of a sequential retrieval to see if that method is acceptable. If it is not, then you can extract a subset of view descriptor data into a SAS data set (or define a DATA step view) using eligible selection criteria, then subset the data set using an application task to achieve the desired performance gains.

If needed, you can mix all of the filtering methods. For example,

```
data work.subset;
  set vlib.imsview; /*View can contain subset criteria*/

  where (additional eligible conditions for IMS SSAs);
  if (ineligible criteria that would not generate SSAs);
run;
```

For all methods, it is possible that a change in criteria can cause an application that once produced SSAs to no longer produce them and resort to using a sequential access method. You can prevent this from happening with the SAS system option IMSWHST=Y. IMSWHST= is an invocation option that can be placed in the restricted options table so that it cannot be changed or overridden. Should the engine detect that no SSAs can be generated when this option is in effect, it will issue a message to the SAS log and terminate the executing task.

Writing Efficient WHERE Statements

Specifying a WHERE statement from which the IMS-DL/I engine can generate SSAs improves performance. The IMS-DL/I engine returns to the SAS System only those database segments that meet your selection criteria. If the IMS-DL/I engine cannot generate SSAs, all segment occurrences for each IMS record (as defined by the path of segments in the view descriptor) are returned to the SAS System for further processing.

To determine whether SSAs are being generated by your WHERE statement, set the option IMSDEBUG=Y or set the number of calls for which you want debugging information.

To ensure that your WHERE statements generate SSAs, do the following:

- When creating descriptors, specify a search field name for all variables you plan to include in your application's WHERE statements, when possible.
- Use one of the eight operators supported by IMS-DL/I in your WHERE statements. The eight operators supported by IMS-DL/I are listed in the following table, along with their alternate forms.

Operator	Alternate Form
=	= or EQ
>	> or GT
<	< or LT
>=	=> or GE
<=	=> or LE
¬=	=¬ or NE
&	* or AND (dependent AND)
|	+ or OR (logical OR)

* Pad the =, >, and < operators with blanks on the right or left.

The ability of the IMS-DL/I engine to generate SSAs also depends on the database type and on the operators that you use in your WHERE expression.

- For GSAM databases, no SSAs can be generated.
- For other database types, the following rules apply:
  - SSAs are generated only for WHERE expressions that involve a variable, an operator, and a literal value. Multiple expressions that use Boolean operators are also allowed. For example:

```
where partnum > 1000
where partnum > 1000 and
                orddate = '31JAN94'd
```

  - The following operators generate SSAs: = (EQ), > (GT), < (LT), >= (GE), <= (LE), IN, BETWEEN, IS NULL, and IS MISSING. For HDAM databases, only the equals (=), IS MISSING, and IN operators generate SSAs.
  - Compound expressions generate SSAs, except when the expressions are joined by OR and the fields involved are in different segments.

For a more detailed description of how WHERE statements work, see "WHERE Statement Processing" on page 142.

Identifying Inefficient SAS WHERE Conditions

When your view descriptor

- uses WHERE clauses that have multiple values for a search field, and

- specifies a path that does not originate from the root segment in the IMS-DL/I database

the view descriptor forces the IMS-DL/I engine to reposition itself to the beginning of the IMS-DL/I database for each value.

In this example, the WHERE statement tries to find two checking account records in the ACCTDBD database.

```
where chckacct = '345620145345'
   or chckacct = '345620134663';
```

Because the CUSTOMER segment is the root segment and the CHCKACCT segment is a child of CUSTOMER, the IMS-DL/I engine must issue a GU call for each checking account number that it wants to find. It does this in order to reposition itself at the start of the database. If it used GN calls, it might pass by one of the records because they are not in sequential order.

Specifying multiple values for a search field in a WHERE statement for HDAM IMS-DL/I databases permits the IMS-DL/I engine to create a WHERE key list. The IMS-DL/I engine issues calls that use, at a minimum, the first segment level SSA with a WHERE key list value. When no more data are retrieved from the IMS-DL/I database for a WHERE key list value, a GU call is used to reposition to the beginning of the database and the next WHERE key list value is used. Processing stops when all WHERE key list values have been used.

The following conditions do not allow the IMS-DL/I engine to generate SSAs. They cause all data from the IMS-DL/I database as defined by the view descriptor to be returned to the SAS System for further processing:

- HDAM WHERE statements that use a WHERE key list and an OR operator with another search field or key list in the first segment level of the view descriptor, for example,

  ```
  where custcode in ('24589689' '29834248')
     | state in ('CA' 'VA');
  ```

- an OR between two segment levels.

Identifying SAS WHERE Conditions That Are Not Acceptable to IMS-DL/I

The following examples are SAS WHERE conditions that are passed to the SAS System for further processing.

- arithmetic expressions, for example:

  ```
  where c1=c4*3
  where c4-c5
  ```

- expressions in which a variable or combination of variables assumes a value of 1 or 0 to signify true or false, for example:

  ```
  where c1
  where (c1=c2)*20
  ```

- concatenation of character variables, for example, **`where c2=D2||D3`**.
- LIKE, BETWEEN, CONTAINS, SOUNDS LIKE operators, for example:

  ```
  where lastname=*'SMITH'
  where lastname like 'D_A%'
  ```

- truncated comparison, for example, **`where cl=:abc`**.
- DATETIME and TIME formats, for example:

```
where ctime= '12:00't
where ctime= '01jan60:12:00'dt
```

- comparisons using operators other than equivalence (=) for character variables, for example:

```
where name>'A'
where ssn<='251-09-7384'
```

- comparisons using operators other than equivalence (=) for date variables not in the YYMMDD format, for example, **`where stmtdate>'01JAN01'D`**. STMTDATE has a DB Content of MMDDYY6.
- references to missing values. This includes the period (.) for numeric variables, and the IS MISSING and IS NULL operators.

```
where stmtdate = .(numeric)

where name = (character)
```

- OR requests for conditions in two hierarchical levels of the database, for example, **`where name='Smith' or stmtamt>0`**. In this example, the NAME field is in the root segment, and the STMTAMT field is in a child segment.
- any WHERE statement for a GSAM database, for example, **`where var1<200`**.
- Any reference to a variable that does not have a SEARCH or SEQ field assigned to it in the access descriptor.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ ACCESS® Interface to IMS-DL/I Software: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999. 316 pp.

SAS/ACCESS® Interface to IMS-DL/I Software: Reference, Version 8

ISBN 1–58025–548–5

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

1st printing, October 1999