



## CHAPTER

## 4

## SAS/ACCESS Data Set Options

---

*Introduction* 43  
*SAS/ACCESS Data Set Options* 43

---

### Introduction

This chapter describes the SAS/ACCESS options that you can specify on a SAS data set in the form **SAS/ACCESS-libref.dbms\_table\_name**. A data set option applies only to the data set on which it is specified. A data set option remains in effect for the duration of the DATA step or procedure.

For information about DBMS-specific data set options and details such as default values, refer to your relational DBMS chapter. In general, whenever there are like-named LIBNAME and data set options, the value of the data set option defaults to the value of the LIBNAME option, if not specified. See “SAS/ACCESS LIBNAME Statement” on page 27 for a description of the SAS/ACCESS LIBNAME statement and its options, which can be assigned to a group of relational DBMS tables or views.

---

### SAS/ACCESS Data Set Options

The SAS/ACCESS data set options are as follows:

- “DBCOMMIT=” on page 44
- “DBCONDITION=” on page 45
- “DBCREATE\_TABLE\_OPTS=” on page 46
- “DBFORCE=” on page 46
- “DBGEN\_NAME=” on page 47
- “DBINDEX=” on page 48
- “DBKEY=” on page 49
- “DBLABEL=” on page 50
- “DBNULL=” on page 52
- “DBPROMPT=” on page 53
- “DBTYPE=” on page 53
- “ERRLIMIT=” on page 54
- “NULLCHAR=” on page 55
- “NULLCHARVAL=” on page 56
- “READ\_LOCK\_TYPE=” on page 56

“SASDATEFMT=” on page 57

“UPDATE\_LOCK\_TYPE=” on page 58

In addition to the SAS/ACCESS data set options that are described in this chapter, you can also use several other SAS data set options when you access DBMS data. The following list includes the SAS data set options that can be used with DBMS data. For a complete listing of data set options that can be used with data sets that are composed of SAS data, refer to the *SAS Language Reference: Dictionary*.

CNTLLEV=

DROP=

FIRSTOBS=

IN=

KEEP=

OBS=

RENAME=

WHERE=

*Note:* The REPLACE= data set option is not supported by SAS/ACCESS engines. If this option is specified, the engine supervisor will print a warning message. △

In this example, the DROP= option causes the SAS/ACCESS engine to omit the SALARY column when it reads the MYDBLIB.EMPLOYEES table.

```
libname mydblib db2 ssid=db2 authid=sasdemo;

proc sql;
select *
  from mydblib.employees(drop=salary)
  where dept='ACC024';
quit;
```

---

## DBCOMMIT=

Enables you to issue a DBMS commit statement automatically after a specified number of rows have been processed.

Default value: 1000

See Also: ERRLIMIT=

### Syntax

**DBCOMMIT=*n***

*n*

is an integer greater than or equal to 0.

### Details

DBCOMMIT= affects update, delete, and insert processing. The number of rows processed includes rows that are not processed successfully. Beginning in Version 8, the

default value of DBCOMMIT= is 1000. If you set DBCOMMIT=0, a commit is issued only once after the procedure or DATA step completes.

In the following example, a commit is issued after every 10 rows are inserted:

```
data oracle.dept(dbcommit=10);
    set myoralib.staff;
run;
```

The DBCOMMIT= option overrides the ERRLIMIT= option. This means that a commit can be issued prior to a rollback that is needed by the ERRLIMIT= option.

## DBCONDITION=

**Specifies criteria for subsetting and ordering DBMS data.**

**Default value:** none

### Syntax

**DBCONDITION=**"*DBMS-SQL-query-clause*"

#### *DBMS-SQL-query-clause*

is a DBMS-specific SQL query clause, such as WHERE, GROUP BY, HAVING, or ORDER BY.

### Details

The DBCONDITION= data set option enables you to pass a DBMS-specific SQL selection condition to the DBMS for processing.

When you create a view descriptor, you can use the PROC ACCESS SUBSET statement to specify selection criteria. These criteria are in the form of DBMS specific SQL query clauses, which the SAS/ACCESS engine passes directly to the DBMS for processing. Beginning in Version 7, the DBCONDITION= data set option enables you to perform the same task. When selection criteria are passed directly to the DBMS for processing, performance is often enhanced. The DBMS checks the criteria for syntax errors when it receives the SQL query.

In the following example, the function passed to the DBMS with the DBCONDITION= option causes the DBMS to return to SAS only the rows that satisfy the condition.

```
proc sql;
    create view smithnames as
        select lastname from myoralib.employees
            (dbcondition="where soundex(lastname) = soundex('SMYTHE')" )
            from payroll2)
    using libname myoralib oracle user=sasdemo pw=sasdemo1 path=dbmssrv;

select lastname from smithnames;
```

---

## DBCREATE\_TABLE\_OPTS=

Specifies DBMS-specific syntax to be added to the CREATE TABLE statement.

Default value: none

---

### Syntax

**DBCREATE\_TABLE\_OPTS='DBMS-SQL-options'**

#### 'DBMS-SQL-clauses'

are one or more DBMS-specific clauses that can be appended to the end of an SQL CREATE TABLE statement.

### Details

DBCREATE\_TABLE\_OPTS= enables you to add DBMS-specific clauses to the end of the SQL CREATE TABLE statement. The SAS/ACCESS engine passes the SQL CREATE TABLE statement and its clauses to the DBMS, which executes the statement and creates the DBMS table. DBCREATE\_TABLE\_OPTS= applies only when you are creating a DBMS table by specifying a libref associated with DBMS data.

In the following example, the DB2 table TEMP is created with the value of the DBCREATE\_TABLE\_OPTS= option appended to the CREATE TABLE statement.

```
libname mydblib db2 user=testuser
      pwd=testpass dsn=sample;

data mydblib.temp (DBCREATE_TABLE_OPTS='PARTITIONING
      KEY (X) USING HASHING');
  x=1; output;
  x=2; output;
run;
```

Given this data set option, the following DB2 SQL statement is passed by the SAS/ACCESS engine to DB2 in order to create the DB2 table:

```
CREATE TABLE TEMP (X DOUBLE) PARTITIONING
      KEY (X) USING HASHING
```

---

## DBFORCE=

Specifies whether to force the truncation of data during insert processing.

Default value: NO

See Also: DBTYPE=

---

### Syntax

**DBFORCE=YES | NO**

**YES**

inserts rows into the specified DBMS table, truncating data values that exceed the length of the DBMS column.

**NO**

does not insert rows into the specified DBMS table when data values exceed the length of the DBMS column.

**Details**

The DBFORCE= option is overridden by the FORCE option when it is used with PROC APPEND or when used with the PROC SQL UPDATE statement. The PROC SQL UPDATE statement does not provide a warning before truncating the data.

In the following example, two librefs are associated with ORACLE databases; the default databases and schemas are used and therefore are not specified. In the DATA step, MYDBLIB.DEPT is created from the ORACLE data referenced by MYORALIB.STAFF. The LASTNAME variable is a character variable of length 20 in MYORALIB.STAFF. During the creation of MYDBLIB.DEPT, the LASTNAME variable is stored as a column of type character and length 10 by using DBFORCE=YES.

```
libname myoralib oracle user=tester1 password=tst1;
libname mydblib oracle user=lee password=dataman;

data mydblib.dept(dbtype=(lastname='char(10)')
  dbforce=yes);
  set myoralib.staff;
run;
```

See your DBMS chapter for details on the option values that are supported by your DBMS.

---

**DBGEN\_NAME=**

**Specifies whether to rename columns automatically when they contain characters that SAS does not allow.**

**Default value:** DBMS

---

**Syntax**

**DBGEN\_NAME=**DBMS | SAS

**DBMS**

renames DBMS columns that contain characters that SAS does not allow into valid SAS variable names.

**SAS**

renames DBMS columns that contain characters that SAS does not allow to the format `_COLn`, where `n` is the column number (starting with zero).

## Details

SAS retains column names when reading data from DBMS tables, unless a column name contains characters that SAS does not allow, such as \$ or @. SAS allows alphanumeric characters and the underscore (\_).

If you specify DBGEN\_NAME=SAS, a DBMS column named `dept$amt` is renamed to `_COLn` where *n* is the column number. If you specify DBGEN\_NAME=DBMS, disallowed characters are converted to underscores, so the same column would be renamed `dept_amt`. If a name is converted to a name that already exists, a sequence number is appended to the end of the new name.

This option is intended primarily for National Language Support, notably the conversion of Kanji to English characters because the English characters converted from Kanji are often those that are not allowed in SAS.

*Note:* These rules apply to the SAS system option VALIDVARNAME=V6 and VALIDVARNAME=V7; the argument V7 can be specified for Version 7 and later.  $\Delta$

---

## DBINDEX=

Indicates whether SAS queries the DBMS to find indexes on the specified table.

**Default value:** DBMS specific

**See Also:** DBKEY=

---

## Syntax

**DBINDEX=**YES | NO | *<'>index-name<'>*

### YES

queries the DBMS to find all of the table's indexes and attempts to use indexes on a DBMS table to improve performance.

### NO

does not attempt to use indexes on a DBMS table.

### *index-name*

uses the specified index name to improve performance.

## Details

If you specify DBINDEX=YES in SAS applications, such as PROC SQL and the DATA step, SAS attempts to use indexes on a DBMS table to improve performance.

If you specify DBINDEX=NO, SAS makes no attempt to use indexes on a DBMS table.

If you specify DBINDEX=*index-name*, the specified index name is used by SAS applications to improve performance. The index name must exist and must be correct for the given context.

When you use the DBINDEX= option with the DATA step KEY= option, you must provide the index name as the value for the KEY= option.

In this example, setting DBINDEX=YES allows the DBMS to use the DBMS index, if one is associated with one or more columns in the MYDBLIB.EMPLOYEES table. If

the EMPID column has an index on it, this query could perform significantly better if DBINDEX=YES is specified.

```
libname mydblib oracle user=karin pw=haggis schema=hrdept;

data keyvalues;
  input empid;
  datalines;
1247
4444
3675
...;
run;

proc sql;
select * from keyvalues a,
  mydblib.employees b (dbindex=yes)
  where a.empid=b.empid;
quit;
```

See your DBMS chapter for DBMS-specific details.

See Chapter 7, “Advanced Topics in SAS/ACCESS,” on page 85 for more information about setting this option for performance enhancement.

---

## DBKEY=

**Specifies the column to use as an index.**

**Default value:** none

**See Also:** DBINDEX=

### Syntax

**DBKEY=**(<'>column-1<'> <... <'>column-n<'>>)

#### *column*

is the name of the column that forms the index on the DBMS table.

### Details

When DBKEY= is specified, you provide the names of the columns that form the index. The index may or may not actually exist on the DBMS table. The DBKEY= option is similar to the DBINDEX=*index-name* option except that instead of providing the index name, you provide the column name or names that form the index. SAS uses the specified column name or names as an index by constructing and passing a WHERE clause to the DBMS. DBINDEX= and DBKEY= are mutually exclusive: if you specify them together, SAS overrides the DBINDEX= data set option and the DBINDEX= LIBNAME option.

You can use the DBKEY= option if the index that you need does not exist in the DBMS or if the engine cannot retrieve index information from the DBMS due to insufficient privileges.

The DBKEY= option can be used to improve performance just as indexing can improve performance. With the DBKEY= option, it isn't necessary to have an existing index on the DBMS table. See Chapter 7, "Advanced Topics in SAS/ACCESS," on page 85 for more information on setting this option for performance enhancement.

*Note:* When you use DBKEY= with the DATA step KEY= option, you must specify DBKEY as the value of the KEY= option, as shown in the following examples.  $\Delta$

```
libname mydblib oracle user=karin pw=haggis schema=hrdept;
data keyvalues;
    input empid;
    datalines;
1247
4444
;

data mydata;
    set keyvalues;
    set mydblib.employees(dbkey=empid) key=dbkey;
run;
```

The next example shows two columns specified as keys. These columns are used to simulate a composite index.

```
libname mydblib oracle user=karin pw=haggis schema=hrdept
    pw=testpass;
data keyvalues;
    input empid jobcode;
    datalines;
1247 10
1266 20
;

data mydata;
    set keyvalues;
    set mydblib.employees(dbkey=(empid jobcode))
        key=dbkey;
run;
```

For more information about using keys, indexes, and WHERE clauses, see Chapter 7, "Advanced Topics in SAS/ACCESS," on page 85.

---

## DBLABEL=

**Specifies whether to use SAS variable labels as DBMS column names during output processing.**

Default value: NO

### Syntax

DBLABEL=YES | NO

**YES**

uses SAS variable labels as DBMS column names.

**NO**

uses SAS variable names as DBMS column names.

**Details**

This option is valid only for creating DBMS tables.

In the following example, a SAS data set NEW is created with one variable C1. This variable is assigned a label of DEPTNUM. In the second DATA step, the MYDBLIB.MYDEPT table is created by using DEPTNUM as the DBMS column name. Setting DBLABEL=YES enables the label to be used as the column name.

```
data new;
  label c1='deptnum';
  c1=001;
run;

data mydblib.mydept(dblabel=yes);
  set new;
run;

proc print data=mydblib.mydept;
run;
```

---

**DBMAX\_TEXT=**

determines the length of a very long DBMS character data type that is read into SAS or written from SAS using a SAS/ACCESS engine.

Default value: 1024

---

**Syntax**

**DBMAX\_TEXT=** <<*integer*>>

*integer*

The <<*integer*> can be between 1 and 32,767.

**Details**

DBMAX\_TEXT= is usually used with a very long DBMS character data type, such as the SYBASE TEXT data type or the ORACLE LONG RAW data type.

---

## DBNULL=

Indicates whether NULL is a valid value for the specified columns when a table is created.

Default value: DBMS specific

See Also: NULLCHAR=, NULLCHARVAL=

---

### Syntax

```
DBNULL=(ALL=YES | NO) | (<column-name-1=YES | NO>
  <... <column-name-n=YES | NO>>)
```

#### *column-name*

is a DBMS column in the table that is being created.

#### YES

indicates that a NULL value is valid for the specified columns in the DBMS table.

#### NO

indicates that a NULL value is not valid for the specified columns in the DBMS table.

#### ALL

indicates that the YES or NO value applies to all columns in the table.

### Details

This option is valid only for creating DBMS tables.

If you specify more than one column name, the names must be separated with spaces.

In the following example, the EMPID and JOBCODE columns in the new MYDBLIB.MYDEPT2 table are prevented from accepting null values by using the DBNULL= option. Any subsequent attempt to insert NULL values into these columns will fail.

```
data mydblib.mydept2(dbnull=(empid=no jobcode=no));
  set mydblib.employees;
run;
```

In the next example, all columns in the new MYDBLIB.MYDEPT3 table, except for the JOBCODE column, are prevented from accepting null values.

```
data mydblib.mydept3(dbnull=(ALL=no jobcode=YES));
  set mydblib.employees;
run;
```

Note that the DBNULL= option processes values from left to right, so if you specify a column name twice, or if you use the ALL value, the last value overrides the first value specified for the column.

See your DBMS chapter for details on the option values that are supported by your DBMS.

---

## DBPROMPT=

Specifies whether SAS displays a window that prompts the user to enter DBMS connection information.

Default value: NO

---

### Syntax

DBPROMPT=YES | NO

#### YES

causes SAS to display the prompting window.

#### NO

does not cause SAS to display the prompting window.

### Details

As a data set option, DBPROMPT= is supported only for view descriptors.

*Note:* DBPROMPT= is not supported in DB2 under OS/390.  $\Delta$

In the following example, connection options are specified in the ACCESS procedure. The DBPROMPT= data set option defaults to NO during the PRINT procedure because it is not specified.

```
proc access dbms=oracle;
  create alib.mydesc.access;
  user=testuser;
  password=testpass;
  table=dept;
  create vlib.myview.view;
  select all;
run;

proc print data=myview;
run;
```

In the next example, the DBPROMPT window opens during connection to the DBMS. Values that were previously specified during the creation of MYVIEW are pulled into the DBPROMPT window fields. The user must edit or accept the connection information in the DBPROMPT window to proceed.

```
proc print data=myview(dbprompt=yes);
run;
```

---

## DBTYPE=

Specifies a data type to use instead of the default DBMS data type when SAS creates a DBMS table.

**Default value:** DBMS specific

**See Also:** DBFORCE=

---

## Syntax

**DBTYPE=**(*<column-name-1=<'>DBMS type<'>>*  
*<...<column-name-n=<'DBMS-type<'>>>*)

### *column-name*

specifies a DBMS column name.

### *DBMS-type*

specifies a DBMS data type.

## Details

This option is valid only for creating DBMS tables.

By default, the SAS/ACCESS engine for your DBMS converts each SAS data type to a predetermined DBMS data type when outputting data to your DBMS. When you need a different data type, you can use DBTYPE= to override the default data type chosen by the SAS/ACCESS engine.

In the following example, DBTYPE= specifies the data types that are used when creating columns in the DBMS table.

```
data mydblib.newdept (dbtype=(deptno=
    'number(10,2)' city='char(25)'));
    set mydblib.dept;
run;
```

See your DBMS chapter for more details on the default data types for your DBMS.

---

## ERRLIMIT=

**Specifies the number of errors that are allowed before SAS stops processing and issues a rollback.**

**Default value:** 1

**See Also:** DBCOMMIT=

---

## Syntax

**ERRLIMIT=***integer*

### *integer*

is a positive integer that represents the number of errors after which SAS stops processing and issues a rollback.

## Details

SAS calls the DBMS to issue a rollback after the specified number of errors occurs during the processing of inserts, deletes, updates, and appends. If ERRLIMIT= is set to

0, SAS processes all rows, regardless of the number of errors that occur. SAS displays the total number of rows processed and the number of failed rows, if applicable, in the log.

If you use the DBCOMMIT= option, DBCOMMIT= overrides the ERRLIMIT= option.

*Note:* If you specify a value for DBCOMMIT= other than zero, rollbacks affected by the ERRLIMIT= option might not include records that are processed unsuccessfully because they were already committed by DBCOMMIT=.  $\Delta$

In the following example, SAS stops processing and issues a rollback to the DBMS at the occurrence of the tenth error.

```
data mydata;
  set mydb.dept(errlimit=10);
  where salary > 40000;
run;
```

---

## NULLCHAR=

Indicates whether a SAS character missing value is inserted into the DBMS column as a NULL value.

**Default value:** SAS

**See Also:** NULLCHARVAL=, DBNULL=

---

### Syntax

NULLCHAR=SAS | YES | NO

#### SAS

indicates that a NULL value is inserted if the DBMS allows NULL. Otherwise, the value that is specified by the NULLCHARVAL= data set option is inserted. If NULLCHARVAL= is not specified, a blank is inserted.

#### YES

indicates that a NULL value is inserted if the DBMS allows NULL. Otherwise, an error is returned.

#### NO

indicates that the value specified by the NULLCHARVAL= data set option is inserted. If NULLCHARVAL= is not specified, a blank is inserted.

### Details

The NULLCHAR= option determines whether or not a NULL is inserted into the DBMS column when there is a character missing value in the SAS data set.

The NULLCHAR= option works in conjunction with the NULLCHARVAL= data set option, which determines what is inserted if NULL values are not allowed. If NULLCHAR=NO, the value specified by NULLCHARVAL= is inserted, regardless of whether the DBMS allows NULLs for the column. NULLCHAR= affects insert and update processing. Note that all SAS numeric missing values (represented in SAS as '?') are inserted into the DBMS as NULLs.

---

## NULLCHARVAL=

Defines the character string to insert into a DBMS table when a character missing value is encountered in a SAS data set or PROC SQL INSERT statement.

**Default value:** a blank character

**See Also:** NULLCHAR=, DBFORCE=, DBNULL=

---

### Syntax

**NULLCHARVAL=**'character-string'

#### character-string

is a string of characters.

### Details

The NULLCHARVAL= option works with the NULLCHAR= option and affects insert and update processing. The NULLCHAR= option determines whether or not a SAS character NULL value is inserted into the DBMS column as a NULL value. If a NULL is not inserted, the value of the NULLCHARVAL= option is inserted. If NULLCHARVAL= is not specified, a blank is inserted.

*Note:* If NULLCHARVAL= is longer than the maximum column width, one of the following occurs on insert:

- The string is truncated if DBFORCE=YES.
- The insert operation fails if DBFORCE=NO.

△

---

## READ\_LOCK\_TYPE=

Specifies how data in a DBMS table is locked during a read transaction.

**Default value:** DBMS specific

**See Also:** UPDATE\_LOCK\_TYPE=  
READ\_LOCK\_TYPE= LIBNAME option on page 39

---

### Syntax

**READ\_LOCK\_TYPE=**ROW | PAGE | TABLE | NOLOCK

#### ROW

locks a row if any of its columns are accessed.

#### PAGE

locks a page of data, which is a DBMS-specific number of bytes.

**TABLE**

locks the entire DBMS table.

**NOLOCK**

does not lock the DBMS table or any rows during a read transaction.

**Details**

If you omit `READ_LOCK_TYPE=`, you get either the default action for the DBMS that you are using, or a lock for the DBMS that was set with the `LIBNAME` statement. You can set a lock for one DBMS table by using the data set option or for all tables in a particular DBMS by using the `LIBNAME` statement option.

If you specify `READ_LOCK_TYPE=TABLE`, you must also specify `CONNECTION=UNIQUE`, or you will receive an error message. Setting `CONNECTION=UNIQUE` ensures that your table lock is not lost, for example, due to another table closing and committing rows in the same connection.

See your DBMS chapter for details on the option values supported for your DBMS.

**SASDATEFMT=**

Changes the SAS date format of a DBMS column.

Default value: DBMS specific

**Syntax**

```
SASDATEFMT=(<DBMS-date-col-1='SAS-date-format'>
  ... <DBMS-date-col-n='SAS-date-format'>>)
```

***DBMS-date-col***

the name of a column in a DBMS table that contains dates.

***SAS-date-format***

a SAS date format that has an equivalent (like-named) informat. For example, `DATETIME21.2` is both a SAS format and an informat, so it can be used as an argument of `SASDATEFMT=`.

**Details**

`SASDATEFMT=` changes the default SAS date format that is assigned to DBMS date values to a new SAS date format that you specify. The default SAS format is based on the data type of the DBMS column.

This option is used to convert DBMS date values to the correct SAS `DATE`, `TIME`, and `DATETIME` values during input operations, to convert SAS date values to the correct DBMS date values during output operations, and to prevent date type mismatches.

For non-English date types, SAS automatically converts the data to the SAS type of `NUMBER`. The `SASDATEFMT=` option does not currently handle these date types, but you can use a `PROC SQL` view to convert the DBMS data to a SAS date format as you retrieve the data, or use a `format` statement in other contexts.

In the following example, the `APPEND` procedure adds SAS data from the `SASLIB.DELAY` data set to the `ORACLE` table that is accessed by

MYDBLIB.INTERNAT. Using SASDATEFMT=, the default SAS format for the ORACLE column DATES is changed to the DATE9. format. Data output from SASLIB.DELAY into the DATES column in MYDBLIB.INTERNAT now converts from the DATE9. format to the ORACLE format assigned to that type.

```
libname mydblib oracle user=karin password=haggis;
libname saslib 'your-SAS-library';
proc append base=mydblib.internat(sasdatefmt=(dates='date9.'))force
  data=saslib.delay;
run;
```

SASDATEFMT= can also be specified with the following alias: SASDATEINFMT=. See your DBMS chapter to determine if your SAS/ACCESS interface supports SASDATEFMT= and, if so, for the default formats that SAS assigns to each DBMS data type.

---

## UPDATE\_LOCK\_TYPE=

**Specifies how data in a DBMS table is locked during an update transaction.**

**Default value:** DBMS specific

**See Also:** READ\_LOCK\_TYPE=

UPDATE\_LOCK\_TYPE= LIBNAME option on page 40

---

### Syntax

UPDATE\_LOCK\_TYPE=ROW | PAGE | TABLE | NOLOCK

#### ROW

locks a row if any of its columns are accessed.

#### PAGE

locks a page of data, which is a DBMS-specific number of bytes.

#### TABLE

locks the entire DBMS table.

#### NOLOCK

does not lock the DBMS row or table during an update transaction.

### Details

If you omit UPDATE\_LOCK\_TYPE=, you get either the default action for the DBMS that you are using, or a lock for the DBMS that was set with the LIBNAME statement. You can set a lock for one DBMS table by using the data set option or for all tables in a particular DBMS by using the LIBNAME option.

See your DBMS chapter for details on the option values that are supported for your DBMS.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ACCESS® Software for Relational Databases: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999. 300 pp.

**SAS/ACCESS® Software for Relational Databases: Reference, Version 8**

Copyright © 1999 by SAS Institute Inc., Cary, NC, USA.

ISBN 1-58025-558-2

All rights reserved. Produced in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

**U.S. Government Restricted Rights Notice.** Use, duplication, or disclosure of the software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227-19 Commercial Computer Software-Restricted Rights (June 1987).

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

1st printing, October 1999

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

The Institute is a private company devoted to the support and further development of its software and related services.