APPENDIX

1

Information for the Database Administrator

Introduction

This appendix explains how the SAS/ACCESS interface to CA-DATACOM/DB works so that you can decide how to administer its use at your site. This appendix also covers system options, performance considerations, debugging, and locking.

How the SAS/ACCESS Interface to CA-DATACOM/DB Works

When you use the ACCESS procedure to create an access descriptor file, the SAS System calls CA-DATADICTIONARY to get a description of the database. When you create a view descriptor file, the SAS System has information about the database in the access descriptor, so it does not call CA-DATADICTIONARY.

The ACCESS procedure writes the descriptor files to a SAS data library. Then, when you use a SAS procedure with a view descriptor whose data are in a CA-DATACOM/DB table, the SAS System Supervisor calls the interface view engine to access the data. The engine can access a CA-DATACOM/DB table for reading, updating, inserting, and deleting. The interface view engine accesses CA-DATADICTIONARY to validate the view descriptor.

When you update either an access descriptor or a view descriptor, the SAS System does not call CA-DATACOM/DB or CA-DATADICTIONARY.

Note: Data records in a CA-DATACOM/DB table cannot be accessed by number. That is, in SAS terms, a CA-DATACOM/DB record is not addressable by row number. Therefore, various SAS procedures behave differently when accessing a CA-DATACOM/DB table than they do when accessing a SAS data file. For example,

- The PRINT procedure issues messages informing you that row numbers are not available and that the procedure has generated line numbers for its output. The numbers do not come from the CA-DATACOM/DB table.
- The FSEDIT procedure does not display a row number in the upper right corner of the window. If you try to enter a number on the command line, an error message appears.

△

How the CA-DATACOM/DB Interface View Engine Works

The interface view engine is an application program that retrieves and updates data in a CA-DATACOM/DB table. Calls are in one of the following categories:

- calls made on behalf of the ACCESS procedure when it is creating an access descriptor
- calls made by a SAS DATA step or by SAS procedures that reference a view descriptor with the DATA= option.

In all situations, the interface view engine initiates and terminates communication between the engine and CA-DATACOM/DB. Each time a different SAS procedure requires use of CA-DATACOM/DB, the program makes an initialization call to the engine. This first call establishes communication with CA-DATACOM/DB. Additional calls to the engine perform retrieval and update operations required by the SAS procedure.

Calls Made on Behalf of the ACCESS Procedure

The ACCESS procedure calls the interface view engine to retrieve information from the CA-DATADICTIONARY database about entity-occurrence names and attributes. The engine sends this information (for example, name, data type, level) to the ACCESS procedure for each field in the table. The procedure stores this information in the access descriptor for later use when creating view descriptors.

Also, if you are using the ACCESS procedure to extract information and place them in a SAS data file, the ACCESS procedure calls the interface view engine.

Calls Made by Other SAS Procedures

SAS procedures can access records in a CA-DATACOM/DB table by referring to a view descriptor with the DATA= option. The SAS System examines the view descriptor to determine which database management system is referred to and passes control to the appropriate engine. The interface view engine uses information stored in the view descriptor (for example, field name, data type, key, level, and occurs specifications) to process CA-DATACOM/DB data records as if they were rows in a SAS data file.

Before doing any retrievals, the engine processes the WHERE clause (if any) to select a subset of data records that are to be processed as rows. The engine constructs the selection criteria from the view WHERE clause and the SAS WHERE clause (if any). If no WHERE clauses exist, all data records in the table qualify.

The interface view engine forms a SAS row (according to the view descriptor), which it passes back to the calling procedure for processing.

Based on the capabilities of the SAS procedure, the next call to the engine might be a request to update or delete the SAS row that was just retrieved. For updates, the engine issues UPDAT, ADDIT, and DELET commands for the data records. Typically the SAS procedure then calls the engine again to retrieve another SAS row. If so, the engine locates another data record, constructs another SAS row, and returns it to the SAS procedure. This cycle continues until the SAS procedure terminates or until the last qualified SAS row has been constructed and returned to the SAS procedure.

Retrieval Processing

Retrievals are done to view data records and also to establish a position for updates and deletions. The type of processing depends on whether you specify a WHERE clause and whether the SORT clause (if any) can be satisfied by simply traversing an index.

Retrievals with a WHERE Clause or SORT Clause

The CA-DATACOM/DB set-at-a-time commands are used for a WHERE clause that can be translated into CA-DATACOM/DB selection criteria. Those commands are also used for a SORT clause that cannot be satisfied by simply traversing an index. The SELFR command builds a Select File according to the WHERE clause and/or SORT clause. Then the SELNR command moves left and right along the Select File, one position at a time. SELNR can also skip directly to the first or last record on the Select File.

The SELSM command skips directly to a specific record that is not adjacent to the current record and that is also not the first or last record on the Select File. Information in the internal record ID (RID) permits the SAS procedure to note any position in a file and return directly to it. The RELES command, issued prior to SELSM, drops the previous lock.

Retrievals with No WHERE Clause

If you do not specify any WHERE clause, the type of retrieval processing depends on the type of SORT clause (if any) and on the Default Key being used.

With no WHERE clause and a SORT clause (if any) that can be satisfied from a single index, the interface view engine uses CA-DATACOM/DB record-at-a-time commands to retrieve data. If you specify a Default Key or let the Default Key default to the Native Key or request ordering that is represented by an index, the interface view engine traverses the respective index for that key.

Note: You can also explicitly set the Default Key to blanks. In this special situation, if you do not specify a WHERE clause or a SORT clause and the CA-DATACOM/DB table is opened for retrieval only, the engine avoids accessing any index; it uses GSETP and GETPS commands to read the data area of the table sequentially in its physical sequence. This method is the fastest way to extract an entire table into a SAS file.

GETPS and GSETP use look-ahead buffering by blocking records. Therefore, to avoid update contention, these commands are used only for retrieval. △

When SAS procedures are doing only retrievals with no WHERE clause and the Native Key is the Default Key, the interface view engine uses GSETL and GETIT commands. These commands do look-ahead buffering by blocking records. For example, when you execute the PRINT procedure with no WHERE clause and the Default Key is the Native Key, the interface view engine

1 opens the table's URT for sequential retrieval.
2 moves low values to the key value portion of the request area and issues the GSETL command for the Native Key. This command rewinds to the beginning of the table.
3 issues GETIT commands until it receives return code 19, which indicates end-of-file.

Note: If you set the Default Key to blanks, PROC PRINT uses GSETP and GETPS commands instead of GSETL and GETIT commands. △

The next example shows how the FSEDIT procedure uses the RDUKG, RDUNX, RDUBR, and RDUKL commands to read and lock records. (These commands are the locking forms of REDKG and so on.) For the FSEDIT procedure, the interface view engine

1 opens the table's URT for direct access with intent to update.
2 reads and locks the first data record by moving low values to the key value portion of the request area and then issuing an RDUKG command for the Default Key (usually the Native Key).
3 scrolls right with the RDUNX command (or left with the RDUBR command). These commands retrieve the next higher (or lower) entry in the index and lock the record, dropping the previous lock.

   If you scroll off the beginning (left) or end (right) of the table, the interface view engine receives an end-of-file signal. In this situation, the engine retrieves the lowest or highest index entry, as indicated, and relocks the data record.

The Internal Record ID (RID)

Occasionally, the interface view engine also uses RDULE and REDLE (locking and not locking) commands, which provide direct addressability. Also, the LOCKG, LOCKL, and LOCNX commands are required when the engine must recover from a situation where another user has deleted a data record that the current user wants to view.

All retrieval commands return an internal record ID (RID). CA-DATACOM/DB sends the RID to the interface view engine. A SAS procedure can request the RID from the engine even though it is internal and not a row number. The engine uses most of the request area for the RID (approximately 256 bytes), not just the internal seven-byte record-ID. Commands such as REDLE and RDULE can use the retained RID information in a rebuilt request area to reestablish the previous position in the index. Thus, after a record is retrieved, its RID can be used to retrieve the record again.

Here is how the FSVIEW procedure uses the RID. For a large table, PROC FSVIEW might need to display several screens of data. FSVIEW asks the engine for the RID for each data record it retrieves and saves the RID. If FSVIEW needs to redisplay the window, it asks the engine to reposition using the specifically saved RID. Once the position is reestablished, FSVIEW can ask the engine to traverse forward or backward to retrieve the records that are needed to fill up the window again.

For example, suppose you are using the FSVIEW procedure to look at a CA-DATACOM/DB table. If you issue the FORWARD command, the engine moves through the entire table and displays the last screen of data. For each new display, FSVIEW notes the RID of the record being displayed at the top of the screen.

If you issue the BACKWARD command to back up a screen, FSVIEW simply asks the engine to point to the RID of the previous screen, rather than reading the table backwards sequentially. The engine issues a RELES command to drop the previous lock, then issues an RDULE command and reads forward one record at a time until it has redisplayed that screenful of data.

If the data record pointed to has been deleted (perhaps by another user), the REDLE/RDULE command fails. In this situation, FSVIEW asks the engine to go forward to the next undeleted data record. Since the engine has saved the RID of the deleted record, it can go forward even though the record itself was deleted.

The LOCKG command allows the interface view engine to position on the first index entry that has the proper key value. Then the engine can move forward with the LOCNX command until it receives an entry with the same key value but a higher internal record-ID or an entry with a higher value than the one requested. The LOCKL command skips backward from a deleted record if the SAS procedure requests it.

Update Processing

Update processing involves updating, deleting, and adding data records. You must retrieve the data record before updating or deleting it.

Adding a new record requires additional processing after the record has been inserted. The position of the new record must be established so the interface view engine can find and display it for interactive online updating. However, a procedure such as the APPEND procedure can avoid the additional processing time for repositioning. If the DDBLOAD= data set option is nonzero, for example, DDBLOAD equals 1, the APPEND procedure loads the data from a SAS file into the CA-DATACOM/DB table with an uninterrupted succession of ADDIT commands. Then you can use a SAS procedure, such as FSEDIT or PRINT, to view the new records.

For more information on the DDBLOAD option, see “Data Set Options” on page 117 and “System Options” on page 115.

Repositioning to an Inserted Record

If the Default Key is the Master Key and DUPE-MASTER-KEY is N, repositioning takes place efficiently. Without a WHERE clause, the RDULE command locks the record. The LOCKX command can locate the index entry just added. Subsequent commands move backward and forward, traversing the actual index.

With a WHERE clause, the interface view engine uses the Compound Boolean Selection (CBS) facility instead of directly traversing a permanent database index. After an ADD, the original Select File does not contain the new record; therefore, it issues a LOCKX command, followed by an RDUID command, which allows the engine to keep an internal table of internal record ID numbers. You can return to the new records without reissuing the WHERE clause.

Repositioning is less efficient if the key value is not guaranteed to be unique. The interface view engine tries to retrieve the newly added record by issuing a SELFR command containing a WHERE clause that comprises all the values in the record just added. If more than one record qualifies, the last record is retrieved, which often is the last record that was added.

In conclusion, here are some external effects of adding new data records.

- For better performance, use the DDBLOAD option to suppress the additional overhead of repositioning. If DDBLOAD equals 1 and there is no WHERE clause, the newly added records are accessible, but you are not repositioned to them when they are added to the table. However, you can find them without leaving the procedure.

  For DDBLOAD=1 and a WHERE clause, the new records are not accessible for viewing since they are not on the original Select File and you suppressed the overhead of keeping track of them. You must either reissue the WHERE clause or exit the procedure and call it again to see the new records.
- If DDBLOAD equals 0 (the default), the engine takes the time to keep track of the new records for repositioning. With no WHERE clause, an entry is placed in the index for the Default Key and that will become the new position. With a WHERE clause or the type of SORT clause that causes a Select File, the engine appends the new record ID at the end of the Select File (not in value order).

Note: Repositioning with a WHERE clause is similar to the SAS base engine processing in which new rows are shown at the end of the SAS data file. On the other hand, repositioning without a WHERE clause reflects the CA-DATACOM/DB processing in which new records are shown in Default Key order. (The record goes in the same place in the table with or without a WHERE clause. This discussion simply explains how it looks to the SAS System.) △

How Changing the CA-DATADICTIONARY Database Affects Descriptor Files

Changes to the CA-DATADICTIONARY database can affect descriptor files. You must fix the descriptors manually if changes to the CA-DATADICTIONARY database invalidate the access or view descriptors. Use the ACCESS procedure to update the access descriptor. Also, update each view descriptor with the ACCESS procedure. You will receive a message if the view descriptor differs from the access descriptor. Change the view descriptor as required.

The interface view engine validates a view descriptor when it opens it. If there is a problem, a message is sent to the LOG window and processing stops. Therefore, you must change the descriptor files manually when changes to CA-DATADICTIONARY invalidate them.

1 When you change the CA-DATADICTIONARY database, you must recreate the access descriptor(s) with PROC ACCESS, using the same name(s).
2 Then you must update each view descriptor with PROC ACCESS. You will get a message if the view descriptor differs from its access descriptor. Change the view descriptor as needed.
3 The SAS/ACCESS interface view engine does a rudimentary validation of a view descriptor when it opens it. For example, it checks the data type information. If it finds a problem, it writes a message to the log and stops.

Before changing CA-DATADICTIONARY, consider the guidelines discussed in the next three sections.

Changes That Do Not Affect Existing View Descriptors

The following changes to the CA-DATADICTIONARY database have no effect on existing view descriptors:

- creating or deleting keys, unless you specified a deleted key as the Default Key.
- adding new fields to a RECORD entity-occurrence.
- deleting fields not referenced in any view descriptor. (Note that if an access descriptor includes the deleted item, users could eventually create a view descriptor using that item, which would be a problem.)
- changing element definitions, as long as a set of one or more elements still exists that can satisfy view descriptors. The interface view engine does not require one element per view descriptor; it looks for the best set of elements for each view descriptor.
- increasing the number of times a field repeats.

Changes That Might Affect Existing View Descriptors

The following changes to the CA-DATACOM/DB database might have an effect on existing view descriptors: changing a field name or decreasing the number of times a field repeats.

Changes That Cause Existing View Descriptors to Fail

The following changes to the CA-DATACOM/DB database cause existing view descriptors to fail:

- inserting or deleting another level in repeating fields.
- changing the attributes of a field. Specifically,
  - You can change the pictures for character fields, but you cannot change them to a numeric type field.
  - You cannot change a numeric data type to a character data type.
- deleting fields that are referenced in a view descriptor.

1

The interface view engine validates the view against the current CA-DATADICTIONARY CA-DATACOM/DB database and issues an informative error message if it detects discrepancies.

SAS System Security

To secure data from accidental update or deletion, you can do the following on the SAS System side of the interface:

- Set up all access descriptors yourself.
- Set up all view descriptors yourself and give them to users on a selective basis.
- Give users read-only access or no access to the SAS data library in which you store the access descriptors. Read-only access prevents users from editing access descriptors and allows them to see only the fields selected for each view descriptor. No access prevents users from doing LIST ALL commands to see the contents of the access descriptor.
- Set up several access descriptors for different users.
- Use the DDBUPD systems option to have a read-only system. (For details, see "System Options" on page 115.)

User Requirements Table (URT)

A User Requirements Table (URT) is a load module that is required by CA-DATACOM/DB. The URT is loaded by the interface view engine and passed to CA-DATACOM/DB when a table is opened. It contains information about how the table is to be accessed. Various values in the URT, such as number and size of buffers, can affect performance.

You can specify a URT in various ways; these are given below. The interface view engine looks for a URT in the order of the situations described. Note that a specific URT always overrides a generic or a default URT.

1 You can designate a specific URT with a data set option when you run a SAS program. For details, see the DDBURT= data set option in “Data Set Options” on page 117.

2 You can designate a specific URT by saving its name in the view descriptor.

3 You can create special URTs named US*tttnnn* or UW*tttnnn* and the interface view engine will look for them in the load library. US means sequential processing, and UW means WHERE clause processing. *ttt* is the CA-DATACOM/DB name for the table, and *nnn* is the database ID. If the interface view engine cannot find such a URT and you have not specified a particular URT, it generates one, as discussed below.

4 You can let the interface view engine create its own URT. It uses two default load modules (named USDDBEXT and UWDDBEXT) that are delivered with the product.

Note: ACCESS=SEQ is not allowed. Use ACCESS=RANSEQ. The engine never alters the type of ACCESS that you specify in a URT. Also, AUTODXC=NO is not allowed in a URT. △

For more information on URTs, see the appropriate CA-DATACOM/DB documentation.

Version 5 URTs work with the Version 8 interface with the following exceptions:

- Version 5 URTs probably have the UPDATE= parameter set to NO. This will fail if you open a view descriptor for update in Version 8. For security reasons, the interface view engine does not upgrade UPDATE=NO to UPDATE=YES. However, the interface view engine does downgrade UPDATE=YES to UPDATE=NO if appropriate in order to prevent possible problems.
- The ACCESS=SEQ parameter in a URT is not supported in Version 8, because additional commands are required to support the engine specifications. Change any existing URTs to ACCESS=RANSEQ.
- The AUTODXC=NO parameter will fail with Version 8. With the SAS locking requests, it tends to exceed CA-DATACOM/DB limits on the number of records locked.

Locks and the Spool File

CA-DATACOM/DB supports record-level locking. It does not support a table lock or any type of member-level locking as in the SAS System. If a procedure requests member-level locking, the interface view engine creates an intermediate file of the SAS records, sometimes called a spool file. This spool file guarantees static data required by the SAS procedure, but at a potentially high processing cost.

A spool file is created if all the following conditions are true:

- The file is opened with member-level locking.
- The file is opened for sequential retrievals.
- The file is opened for a procedure that requires multiple passes or "by-rewinds".

Note: The spool file creates a temporary file of static data. It does not prevent other users from changing the data in the table. △

The processing costs may be so high that some tables cannot be processed. Therefore, a DDBLOCK= data set option is available that instructs the interface view engine not to build the intermediate file. If DDBLOCK equals 1, a warning message appears, but the procedure continues to execute. The user executes the procedure at his own risk. Presumably, that user is the only one using the table or the table is under exclusive use by some method separate from the SAS System.

Alternatively, if you are concerned about keeping the data static while the SAS procedure executes, you could extract the CA-DATACOM/DB data into a SAS data file, then run the procedure against that data file.

Direct Addressing and Access by Row Number

Direct (random) addressing is supported by the SAS/ACCESS interface to CA-DATACOM/DB. However, access by row number is not supported, because qualified records can float around if your updates or other users' deletions move the records out of your WHERE clause context.

For example, if you are on row 3 while someone else is deleting row 4, you go forward to row 5. In another situation, if you update row 3 so that it no longer matches your WHERE clause, it is gone if you ever try to go back to it (even in the same session). CA-DATACOM/DB re-evaluates each retrieval against the WHERE clause and will not return data records that once qualified but currently do not qualify.

Password Encryption/Decryption

The CA-DATADICTIONARY password is encrypted and decrypted with SAS routines.

Maximizing the Interface Performance

Among the factors that affect the interface performance are the size of the table being accessed, the number of fields being accessed, and the number of data records qualified by the selection criteria. For tables that have many fields and many data records, you should evaluate all SAS programs that need to access the table directly. In your evaluation, consider the following questions:

- Does the program need all the CA-DATACOM/DB fields? If not, create and use an appropriate view descriptor that includes only the fields needed.
- Do the selection criteria retrieve only those data records needed for subsequent analysis? If not, specify different conditions so that the selected records are restricted for the program being used.

- □ Are the data going to be used by more than one procedure in one SAS session? If so, consider extracting the data and placing them in a SAS data file for SAS procedures to use, instead of allowing the data to be accessed directly by each procedure. See "Performance Considerations" on page 74 for circumstances in which extracting data is the more efficient method.
- □ Do the records need to be in a particular order? If so, include a SORT clause in the appropriate view descriptors or a SAS BY clause in a SAS program.
- □ Do the selection criteria allow CA-DATACOM/DB to use key (indexed) fields and non-indexed fields efficiently? See “WHERE Clause in a View Descriptor” on page 91 for some guidelines on specifying efficient selection criteria.
- □ Does your WHERE clause cause CA-DATACOM/DB to create a temporary index, which often requires excessive processing time? For more information on displaying WHERE clauses and messages about creating temporary indexes, see “DDBTRACE= Data Set Option” on page 119.
- □ What kind of locking mechanism is required? (See “Locks and the Spool File” on page 112.)
- □ Are your CA-DATADICTIONARY elements well-matched to your view descriptors?
- □ Would a different URT help?
- □ Would use of a default key give you a faster result?

Multi-Tasking

The SAS System creates a new task for each window that is opened by each user. If the host option SYNCHIO equals NO (the default for some environments), asynchronous processing can occur. That is, work in multiple windows can be done concurrently.

CA-DATACOM/DB also supports concurrent tasking. The interface view engine uses Option 3 (the most flexible one), described in the chapter called "Extended Programming" in the CA-DATACOM/DB *System Programming Guide*. When the SAS System creates a new task, the interface view engine also creates a new task to communicate with CA-DATACOM/DB.

At initialization time, CA-DATACOM/DB must know the maximum number of concurrent tasks that a particular address space will have active. Use the DDBTASK system option to specify this number to the interface view engine. The default is 2. The engine will reject attempts to open more than the maximum number of tasks specified and will issue an error message

For a given address space, the CA-DATACOM/DB operator display facility typically shows a given SAS address space as having one user active for CA-DATADICTIONARY communications and one user active for database work. One task is allocated to CA-DATADICTIONARY, and the number of tasks specified in DDBTASK is allocated to database work. For example, suppose you specify DDBTASK equal to 4. The CA-DATACOM/DB operator facility would allocate and display a total of five tasks for your copy of the SAS System.

Error Messages and Debug Information

If you are experiencing a problem with the SAS/ACCESS interface to CA-DATACOM/DB, the technical support staff at SAS Institute Inc. might ask you to provide additional debug information. They may instruct you to set a debugging option for your job and rerun it.

The DDBTRACE option is available as a data set option on your PROC statement or DATA step. You can also set DDBTRACE as a systems option. For example, if DDBTRACE equals 1, you can look at the WHERE clause that was processed. If you specify DDBTRACE=1, the view descriptor WHERE clause and the SAS WHERE clause, if any, appear on the SAS log. The display also shows the various fields that make up the key fields. For example, suppose POLI is a key field composed of two fields, PO and LI, and you specify

```
poli eq \2222\0000
```

The WHERE clause display echos back

```
po eq 2222 and li eq 0000
```

Note: The engine translates the SAS WHERE clause (if any) as much as possible into CA-DATACOM/DB format and connects it to the view WHERE clause with the AND operator. △

The text of most error messages is self-explanatory. However, some error messages have a prefix containing a display code decimal number. This prefixed number contains the CA-DATACOM/DB return code and the internal return code. For example, in the following message

```
10.0 Duplicate Master Key not allowed
```

the number 10 is the return code and 0 is the internal return code.

System Options

SAS system options for the CA-DATACOM/DB interface are specified the same way as other SAS system options. The CA-DATACOM/DB options are invocation options. Therefore, you can specify the options in a configuration file, in the DFLTOPTS table, or when you invoke the SAS System. They cannot be changed during the SAS session.

Several system options (for example, DDBDBN=, DDBPW=, DDBUSER=, and DDBSV=) can also be specified as data set options. A data set option value can override a system option for the duration of a single procedure or DATA step. See “Data Set Options” on page 117 for more information about data set options.

Several system options control the default values used when creating a new access descriptor. You can override the default values by specifying different values in the ACCESS procedure or by setting the appropriate values to the options. Here are some examples of setting system options for the interface:

```
DDBDBN=INVENTORY
DDBLOAD=1
```

The first system option sets INVENTORY to be the CA-DATACOM/DB database name. The second system option requests the CA-DATACOM/DB engine to keep track of the number of inserts to the database.

Another useful system option is DDBUPD. This option specifies whether the interface view engine is allowed to perform updates against the CA-DATACOM/DB tables. The value Y allows updates; the value N allows read-only access. When the value is N, any attempt to update a CA-DATACOM/DB table is rejected and an error message is written to the SAS log. The default value is Y.

Note: In previous releases of SAS/ACCESS interface to CA-DATACOM/DB, DDBUPD was called DDBENGMD. △

Once your SAS session is executing, you can display the values of system options by entering the following SAS statements:

```
proc options ddb;
run;
```

The system options for the SAS/ACCESS interface to CA-DATACOM/DB are written to the SAS log. Note that you cannot see the options for any passwords.

For convenience, you may want to set certain options during installation rather than with each SAS invocation. In addition, you may want to restrict certain options so they cannot be changed for a SAS session. You do this by specifying their values in the Restricted Options Table during installation. Refer to the installation instructions for details.

The CA-DATACOM/DB system options are listed in Table A1.1 on page 116.

Table A1.1 CA-DATACOM/DB System Options

Systems Option	Default	Purpose
DDBDBN	blanks	Database name
DDBPW	blanks	Password for CA-DATADICTIONARY
DDBSV	PROD	Status/Version
DDBURT	blanks	User Requirements Table to be used
DDBUSER	blanks	Userid for CA-DATADICTIONARY
DDBDELIM	\	Changes the delimiter
DDBUPD (formerly DDBENGMD)	0	Engine mode (update or read-only)
DDBLOAD	0	Mode for loading data records
DDBLOCK	0	Spooling mechanism
DDBMASK	#	Changes the mask character
DDBMISS	blank	Sets missing values to blanks or X'00
DDBSPANS	*	Changes the SPANS character
DDBTASK	2	Number of concurrent tasks
DDBTRACE	0	Displays WHERE clauses and debug traces

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ACCESS Interface to CA-DATACOM/DB Software: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999. pp. 170.

SAS/ACCESS Interface to CA-DATACOM/DB Software: Reference, Version 8

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

ISBN 1–58025–545–0

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

1st printing, October 1999