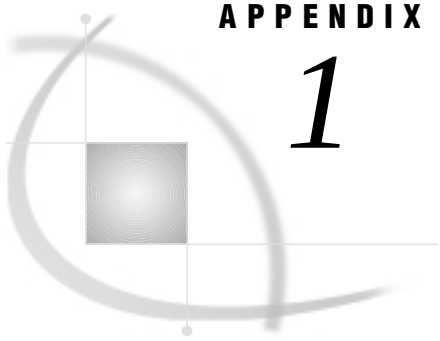

APPENDIX

1

Information for the Database Administrator

Introduction

This appendix explains how the SAS/ACCESS interface to ADABAS works so that you can decide how to administer its use at your site. This appendix also discusses the effects of changing ADABAS data on SAS/ACCESS descriptor files, data security, controlling data locks, maximizing the ADABAS interface view engine performance, how to debug problems, and defaults for systems options.

How the SAS/ACCESS Interface to ADABAS Works

When you use the ACCESS procedure to create a SAS/ACCESS access descriptor file, the SAS System calls ADABAS to get a description of the ADABAS data. When you create a view descriptor file, the SAS System has information about the ADABAS data in the access descriptor, so it does not call ADABAS.

The ACCESS procedure writes the SAS/ACCESS descriptor files to a SAS data library. Then, when you issue a SAS procedure using a view descriptor whose data are in an ADABAS file, the SAS System Supervisor calls the interface view engine to access the data. The engine can access ADABAS data for reading, updating, inserting, and deleting.

When you edit either an access descriptor or a view descriptor, the SAS System does not call ADABAS.

ADABAS data records are uniquely identified by an Internal Sequence Number (ISN). As discussed in Chapter 2, "ADABAS Essentials," on page 7, multiple SAS observations are generated from a single ADABAS record when the view descriptor contains periodic group fields. Creating multiple SAS observations does not preserve the unique quality of the ISN number (that is, more than one SAS observation can refer to a single ADABAS record). As a result, ADABAS records cannot be uniquely addressed by a single number within the SAS environment.*

In SAS terms, this means that an ADABAS record is not addressable by an observation number. Therefore, various SAS procedures behave differently when accessing ADABAS data than they do when accessing a SAS data file. For example, consider the following PRINT procedure and FSEDIT procedure behavior with ADABAS data:

- The PRINT procedure issues messages informing you that observation numbers are not available and that the procedure has generated line numbers for its output. The numbers do not come from the ADABAS file.
- The FSEDIT procedure does not display an observation number in the upper right corner of the window. If you try to enter a number on the command line, an error message is displayed.

How the ADABAS Interface View Engine Works

The ADABAS interface view engine is an applications program that retrieves and updates ADABAS data. 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 SAS System and ADABAS. Each time a different SAS procedure requires use of ADABAS, the program makes an initialization call to the engine. This first call establishes communication with ADABAS. Additional calls to the engine perform retrieval and update operations required by the SAS procedure.

* In combination, the SAS variables that contain the ISN and periodic group occurrence number uniquely identify an observation. The periodic group occurrence number variable is a fabricated SAS variable that does not have a corresponding field in the ADABAS file. It can be selected when creating a view descriptor and is valued with the occurrence number of each periodic group accessed.

Calls Made on Behalf of the ACCESS Procedure

For both NATURAL DDMs and ADABAS files, the ACCESS procedure calls the interface view engine to retrieve data field information. The engine sends this information (such as, name, level number, data format, and definition options) to the ACCESS procedure for each ADABAS data field.

When you specify a DDM name, the interface view engine retrieves information from two places. First, the engine uses a combination of S1 and L1 commands to search and retrieve the DDM records that have been previously cataloged into a system file. The DDM records contain information for each field included in the DDM. Along with the field information, the engine also obtains the ADABAS file number and the database identifier on which the DDM is based. The ADABAS file number and database identifier are used in conjunction with the LF command to retrieve even more information directly from the Field Definition Table (FDT). The engine then combines the information retrieved from the DDM and the FDT to give a detailed description of each field.

When dealing directly with an ADABAS file, the engine uses only the LF command for retrieving field information from the FDT. The ACCESS procedure stores this information in the access descriptor file for later use when creating view descriptors.

If you use the ACCESS procedure to extract data and store them in a SAS data file, the ACCESS procedure calls the interface view engine to retrieve the actual data.

Calls Made by Other SAS Procedures

SAS procedures can access records in an ADABAS file 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 (such as name, level number, data format, and definition options) to process ADABAS data records as if they were observations in a SAS data file.

Before doing any retrievals, the engine processes the WHERE clause (if any) to select a subset of data records to be processed as observations. The engine inspects the view WHERE clause and the SAS WHERE clause (if any) and issues the ADABAS commands that are necessary to qualify the appropriate records. If no WHERE clause exists, all data records in the file qualify.

The interface view engine forms a SAS observation (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 observation that was just retrieved. For updates, the engine issues reads with holds followed by the appropriate update command. Adds do not require a record to be read (except in special cases when you are dealing with ADABAS files that contain periodic group fields).

The SAS procedure then calls the engine again to retrieve another SAS observation. The engine locates another data record, constructs another SAS observation, and returns it to the SAS procedure. This cycle continues until the SAS procedure terminates or until the last qualified SAS observation has been constructed and returned to the SAS procedure.

Retrieval Processing

This section discusses retrieval processing. The type of processing and the subset of ADABAS commands used by the interface view engine depends on

- whether you specify a WHERE clause (view or SAS) and sorting criteria (view SORT clause, SAS BY statement, SAS ORDER BY clause) separately or in combination

- whether the SAS procedure requires sequential or random access of the ADABAS records
- whether the SAS procedure requests exclusive control of the ADABAS data, either implicitly or explicitly (using the SAS software CNTLLEV=MEMBER data set option).

Retrievals with No WHERE Clause and No Sorting Criteria

If you do not specify a WHERE clause or any sorting criteria, the type of ADABAS commands used for retrievals is controlled by the type of access (either sequential or random) required by the SAS procedure. A SAS procedure requiring sequential access (for example, PROC PRINT) results in the engine issuing L2 commands to retrieve the records from the ADABAS file. Since there is no WHERE clause to subset the data, the engine retrieves every ADABAS record.

A SAS procedure requiring random access (for example, PROC FSEDIT) must have the ability to navigate both forward and backward. To support forward and backward navigation, an ISN list must exist. When a WHERE clause has not been entered, the engine generates a default WHERE clause. The engine searches for the first ADABAS descriptor data field in the view descriptor. Once the engine finds an ADABAS descriptor field, its format and length are used to construct a default WHERE clause. (If no ADABAS descriptors exist, the engine displays an error message.)

The ADABAS field formats and their corresponding default WHERE clause are listed below (assuming that the data field named AA is the first ADABAS descriptor field):

Format	Default WHERE Clause
alphanumeric	where aa >= '*b*'
binary	where (aa <= 0) or (aa > 0)
fixed point	
floating point	
packed decimal	
unpacked decimal	

The default WHERE clause results in the ADABAS interface view engine issuing S1 and S8 commands. Those commands generate an ISN list whose corresponding records are read using L1/L4 commands. The engine uses L4 commands if the SAS procedure is capable of performing updates (that is, PROC FSEDIT). The engine uses L1 commands if the SAS procedure is not allowed to perform updates (that is, PROC FSBROWSE).

Note: A default WHERE clause can use considerable resources, depending on the number of ADABAS records. Therefore, for large amounts of ADABAS data, it is best to include either a view WHERE clause or a SAS WHERE clause. Also, the ADBDEFW systems option and ADBL3 data set option are available to alter the interface view engine's handling of the default WHERE clause. A default WHERE clause may also be issued for an ADABAS descriptor that has the NULL SUPPRESS option. That is, ADABAS records may exist that are not pointed to by the ISN list. △

Retrievals with Only a WHERE Clause

If you specify a WHERE clause (either view or SAS), the engine typically issues S1, S8, and L1/L4 commands to extract the appropriate ADABAS records. The only instance where this does not apply is when the L3 command is used. (This case is discussed later.)

If you specify both a view WHERE clause and a SAS WHERE clause, the two are combined using the Boolean AND operator, that is,

(SAS WHERE clause) AND *(view WHERE clause)*

Note: The only part of the SAS WHERE clause being logically combined is the part that ADABAS can support. See "Using a SAS WHERE Clause for Selection Criteria" on page 132. △

Combining the two WHERE clauses does not alter the set of commands used to retrieve the records. It does require the execution of an additional S8 command. The S1 and S8 commands generate an ISN list whose records are subsequently read using L1/L4 commands.

To optimize WHERE clause processing, you can specify use of the L3 command with the SAS software ADBL3 data set option. The ADBL3 data set option also controls which commands are used if the L3 command cannot be used. A number of restrictions must be satisfied before the L3 command can be used.

- The SAS procedure must have exclusive control of the view descriptor and therefore exclusive control of the underlying ADABAS data. This control is accomplished implicitly by some procedures and explicitly by using the SAS software CNTLLEV=MEMBER data set option.
- The SAS procedure must request sequential access.
- Sorting criteria cannot be specified.
- A WHERE clause can contain only a single condition.
- The field referenced in the single condition must be an ADABAS descriptor field. (Phonetic descriptors and descriptors contained within or derived from a field within a periodic group cannot be used.)
- The operator used in the single condition must be LT, LE, GT, GE, or SPANS.

The L3 command reads data records in logical sequential order based on the sequence of values for a given ADABAS descriptor field. The inverted list associated with the descriptor field controls the order in which the records are read. Unlike the S1 command that creates an ISN list, the L3 command uses an existing inverted list resulting in more optimal retrievals. The L3 command produces the most dramatic results for very large ADABAS files, or in ADABAS environments where ISN list buffer sizes are set comparatively low, or in system environments where disk space is a problem.

If the L3 command cannot be used, the ADBL3 data set option lets you specify the use of either the S1 or the S2 command to retrieve data records in its place. If the S2 command cannot be used, the engine returns an error.

Retrievals with Sorting Criteria

To sort data records, you can use only ADABAS descriptor fields since both ADABAS commands used for sorting rely on ADABAS descriptors. The S9 command requires an ISN list as input, and the L3 command uses an inverted list. This means that all ADABAS data fields referenced in a view descriptor SORT clause, a SAS BY statement, or a SAS ORDER BY clause must be associated with ADABAS descriptor fields.

As with the WHERE clause, certain sorting criteria can be optimized with the L3 command. However, the following conditions must apply before the L3 command can be used for sorting:

- The SAS procedure must request sequential access.
- Only one sort field is requested.
- A WHERE clause cannot be specified.
- The sorting sequence must be ascending.

You invoke the L3 command with the SAS software ADBL3 data set option. The L3 command reads data records in logical sequential order using the inverted list associated with the ADABAS descriptor field. The inverted list is maintained in ascending logical order.

If the ADBL3 data set option is not set, or it specifies use of the L3 command only and one of the above conditions is not met, the S9 command is used to satisfy the sorting criteria. The S9 command also imposes some limitations: a maximum of three descriptor fields can be used for sorting, and the ordering sequence (either ascending or descending) applies to every sort field. In all cases, the S9 command requires an ISN list as input. Since the ISN list is generated by WHERE clause processing, a default WHERE clause must be used if a WHERE clause is not specified. The S9 command generates a final ISN list in sorted order. L1/L4 commands are used to read the ADABAS records represented in the final ISN list.

The S9 command can also sort the input ISN list in ascending ISN sequence. This is accomplished by supplying only the ordering verb ASCENDISN (no sort fields) in the view descriptor SORT clause.

Update Processing

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

Updating, deleting, and adding records is a straightforward process if there are no periodic group fields in the view descriptor or in the ADABAS data on which the view descriptor is based. In this case, the A1, E1, and N1 ADABAS commands are used for updating, deleting, and adding records, respectively.

If periodic groups do exist, adding new records and deleting existing records is more complicated. This is due to multiple SAS observations being generated from a single ADABAS record containing periodic group fields. The complexities of adding records containing periodic group fields is discussed in “Adding an Observation” on page 122. Deleting records when the view descriptor or ADABAS data contains periodic group fields is discussed in “Deleting an Observation” on page 121.

Competitive Updating and Logical Transaction Recovery

The interface view engine is an ET logic user application program. The ET (End Transaction) command and the record HOLD facility manage disaster recovery and multi-user concurrency issues.

SAS procedures capable of performing updates use the L4 command (read data record with hold) to read and hold data records. The held record is released with an ET command just before the next record is read. This means that any system or program failure recovers updates up to, but not necessarily including, the last ADABAS record read. When processing ADABAS data with periodic groups, remember that many SAS observations may represent one ADABAS record. Therefore, it is possible to have updated several SAS observations without issuing an ET command.

If an update procedure requests a record that another update procedure has locked, the read fails. The interface view engine recognizes this condition and re-issues the read without the HOLD option. The record is displayed with a message indicating that the record was unable to be locked and cannot be updated.

SAS procedures that do not have update authorization use the L1 command when reading records. The L1 command does not place the record in hold status, and subsequent ET commands are unnecessary.

Effects of Changing an ADABAS File or NATURAL DDM on Descriptor Files

Changes to an ADABAS file or NATURAL DDM can affect associated SAS/ACCESS descriptor files. If changes to ADABAS data invalidate your descriptor files, you must fix them manually by following these steps:

1. When you change an ADABAS file or NATURAL DDM, you must re-create the access descriptor(s) with the ACCESS procedure, using the same access descriptor name(s).
2. Then you must update each view descriptor with PROC ACCESS. Change the view descriptor as needed.
3. The SAS/ACCESS interface view engine does a rudimentary validation of a view descriptor upon opening it. For example, the engine checks data type and data field grouping information. If a problem is found, the engine writes a message to the log and stops.

Before changing ADABAS data, consider the guidelines listed below.

Changes That Have No Effect on Existing View Descriptors

The following changes to an ADABAS file or NATURAL DDM have no effect on existing view descriptors:

- creating ADABAS descriptors.
- inserting new data fields.
- deleting data fields not referenced in any view descriptor. (Note that if an access descriptor includes the deleted data field, users could eventually create a view descriptor using that data field, which would be a problem.)

Changes That Might Have an Effect on Existing View Descriptors

The following changes to an ADABAS file or NATURAL DDM might have an effect on existing view descriptors:

- changing a data field name. If the data field name was used in selection criteria stored in the view descriptor, when you try to use the view descriptor, you will receive a syntax error message indicating an unrecognized data field name.
- deleting ADABAS descriptor data fields if the field is used in selection criteria.

Changes That Cause Existing View Descriptors to Fail

The following changes to an ADABAS file or NATURAL DDM cause existing view descriptors to fail when they are used:

- changing a numeric type to a character type.
- changing a character type to a numeric type.
- deleting a data field that is referenced in a view descriptor.
- modifying a periodic group field or a multiple-value field. A field defined as a periodic group, a field within a periodic group, or a multiple-value field must retain its properties.

- changing lengths that affect a SAS format. Certain ADABAS numeric types are changed to character hexadecimal when their lengths are too large for the SAS System to handle. You cannot change lengths that result in changing SAS formats from character to numeric or numeric to character.
- changing superdescriptors, superfields, subdescriptors, and subfields. You cannot change the definition of these field types, such as adding or subtracting parentage information, changing the order of parentage information, or changing the from-to specification.

Data Security

The SAS System preserves data security provided by ADABAS, NATURAL, and your operating system. As the DBA, you have control over who has security. You control who can create ADABAS files, and creators of the files control who can access the data. Therefore, SAS System users can access only ADABAS files they created or ones for which they have been granted specific security options.

To secure data from accidental update or deletion, you can take precautionary measures on both sides of the interface view engine.

How the Interface View Engine Uses Security Specifications

This section contains an explanation of how the interface view engine works in conjunction with both ADABAS Security and the NATURAL SECURITY System (NSS).

The twelve ADABAS Information fields and the three NSS fields discussed in this section can be passed to the interface view engine via a view descriptor or through the use of data set options. For simplicity, it is assumed that each field has been stored in a view descriptor.

If ADABAS Security is in use at your site, up to four separate fields of information may need to be provided for each of three ADABAS files that the interface view engine may reference during the execution of a single SAS procedure. The three ADABAS files are

- the file from which data records are to be retrieved and updated (generally referred to as the ADABAS file)
- a system file containing DDM information
- a security file containing NSS information about the applications and users defined at your site.

The four fields of information associated with each ADABAS file are

- file number
- password
- cipher code
- database identifier.

The file number is the number of the ADABAS file from which data records are actually retrieved and updated. The database identifier field indicates which database contains the file. The interface view engine obtains the file number and database identifier from one of two places: either directly from a view descriptor or from a DDM.

If you want to work directly with an ADABAS file (without referencing a DDM), you must supply the file number and database identifier in the view descriptor. (If the ADABAS file is protected using ADABAS Security, you must also supply a password and/or cipher code in the view descriptor.) The engine reads the file number and

database identifier from the view descriptor and uses them to retrieve and update the appropriate records.

If a DDM name is stored in the view descriptor, the engine obtains the ADABAS file number and database identifier from the DDM. Since a DDM is stored as one or more records in a system file*, the engine must read that file to obtain the file number and database identifier.

The system file containing the DDM records is just another ADABAS file. As such, it has a corresponding database identifier and can be security-protected using ADABAS Security. If the system file is protected, you must supply the necessary password and/or cipher code in the view descriptor. The database identifier must also be stored in the view descriptor.

Once the file number and database identifier are obtained from the DDM, the engine retrieves and updates the appropriate records.

The security file is the third and final ADABAS file that may be referenced by the interface view engine. The security file contains the NATURAL SECURITY data (for each user identifier, application identifier, file, and so on) that are defined at your site.

The interface view engine requires that the security file number, password, cipher code, and database identifier be provided in these situations:

- NSS is installed at your site.
- The view descriptor refers to a DDM name.
- The DDM referenced in the view descriptor has been defined to NSS.
- A library identifier, user identifier, and password have been supplied to the engine via the view descriptor or through data set options.

(The security file password and cipher code are required only if the security file has been security-protected using ADABAS security.)

In order to communicate with NSS, the interface view engine needs the security file number, password, cipher code, and database identifier, as well as the NSS library identifier, user identifier, and user password. The engine can use only library identifiers and user identifiers that were previously identified to NSS.

An application program must determine whether a specific library identifier and user identifier have authorization to access or update a particular DDM. To do that, Software AG developed an interface to NSS, which is delivered as a load module named NSCDDM.

The interface view engine uses this NSS interface to check access/update authorization for a library identifier and user identifier. If they do not have the appropriate authorization, an error message is displayed.

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. Drop data fields containing sensitive data from display by using the DROP statement.
- Set up all view descriptors yourself and give them to users on a selective basis. You can store the appropriate security options in the SAS/ACCESS descriptors, or you might not want to store any options so that users must supply security with data set options. You can also include view WHERE clauses to restrict data access.

* Using the NATURAL View Maintenance application (SYSDDM) is one method of defining and cataloging a DDM in a system file. Your site may have more than one NSS system file.

- Give users read-only access or no access to the SAS data library where you store the access descriptors. Read-only access prevents users from editing access descriptors and allows them to see only the data fields selected for each view descriptor.
- Set up several access descriptors for multiple security options, or require the user to create them.
- Set the ADBUPD systems option to R (for read only) to disable all updates from the SAS System. See "ADBEUSE Systems Options Default Values" on page 116.

ADBSE User Exit

The SAS/ACCESS interface also provides a user exit named ADBSE for execution and access authorization. For information regarding this user exit, contact your SAS Technical Support representative.

Effects of Changing Security Options

The owner of an ADABAS file or NATURAL DDM can change any security option at any time. If a security option that is stored in a view descriptor is changed, you can either update the view descriptor or override the stored option each time you need to use the view descriptor. The software does not require that you use the same option, but either option must have enough authority to service the view descriptor.

Security options can be stored in access descriptors or associated view descriptors. However, changing a security option does not affect access descriptors or view descriptors. The access descriptor still has all its data fields, but you might not be able to use the view descriptor.

Note that if an access descriptor was created with Assign Security=YES, data set options cannot override security specifications included in a SAS/ACCESS view descriptor.

Controlling Data Locks

SAS software supports several levels of data locking, which is a means of holding information constant so that it doesn't change unexpectedly. The control level is the degree to which a SAS procedure can restrict access to data. SAS procedures can request locks on individual records, on library members, and so on. Locking is also controlled by the SAS software CNTLLEV data set option, which can request record-level locking and member-level locking. Some SAS procedures set CNTLLEV equal to MEM internally for their own processing reasons. Many statistical procedures must make multiple passes of the data. For example, finding the median requires more than one pass.

The ADABAS interface view engine honors all locking requests in a multi-user environment. (Locks are not required in a single-user environment.) The following conditions apply:

- If there are no locking requests, you cannot update ADABAS data.
- For record-level locking, ADABAS locks one ADABAS logical record at a time. If the record contains a periodic group, the lock will include one or more SAS observations.
- For member-level locking, ADABAS puts a hold on the entire ADABAS file.

For more information on how the interface view engine handles locking, see "Competitive Updating and Logical Transaction Recovery" on page 110.

Maximizing ADABAS Performance

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

- Do the selection criteria allow ADABAS to use ADABAS descriptor data fields efficiently? See "Creating and Using View Descriptors Efficiently" on page 98 for some guidelines on specifying efficient selection criteria.
- Does the program need all the ADABAS data fields? If not, create and use an appropriate view descriptor that includes only the data fields to be used.
- Do the selection criteria retrieve only those logical records needed for subsequent analysis? If not, specify different conditions so that the selected data are restricted for the program being used.
- Are the data going to be used by more than one procedure in a single 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 36 for circumstances when extracting data is the more efficient method.
- Do the data need to be in a particular order? If so, include a SORT clause in the appropriate view descriptor or a SAS BY statement in the SAS program.
- What kind of locking mechanism will ADABAS need to use? See "Controlling Data Locks" on page 114.
- Are you using a view SORT clause, a SAS BY statement, or a SAS ORDER BY clause without either a view WHERE clause or a SAS WHERE clause? Without a WHERE clause, the engine qualifies all ADABAS data to be sorted, which can use a considerable amount of resources.

Debug Information for Problems

If you are experiencing a problem with the SAS/ACCESS interface to ADABAS, the Technical Support staff at SAS Institute might ask you to provide additional debug information. They may instruct you to set a debugging option for your job and rerun it.

The ADBTRACE option is available as a data set option on your PROC statement or DATA step. For example, if ADBTRACE=1, you can look at the WHERE clause that was processed. That is, the SAS WHERE clause, if any, and the combined view WHERE clause appear on the SAS log.

The ADBTRACE= data set option can also be used to produce other types of traces for debugging purposes. Contact your SAS Technical Support representative if you need more information.

Systems Options for the ACCESS Procedure and the Interface View Engine

Certain values used by the ACCESS procedure and the interface view engine are stored in two CSECTs (Assembler language constant sections) and automatically linked

with the SAS/ACCESS software load modules as systems options. The two CSECTs are: ADBAUSE for the ACCESS procedure and ADBEUSE for the interface view engine.

The systems options associated with PROC ACCESS control the default values used when creating a new access descriptor. The systems options associated with the interface view engine control various run-time characteristics of the engine.

ADBAUSE Systems Options Default Values

The following systems options set default values to be used by the ACCESS procedure when you create an access descriptor. You can override the default values by specifying different values by using the NSS, ADBFILE, SYSFILE, and SECFILE statements in the ACCESS procedure.

Option	Default	Purpose
ADBNATAP	blanks	NATURAL SECURITY system library identifier.
ADBNATPW	blanks	NATURAL SECURITY system user password.
ADBNATUS	blanks	NATURAL SECURITY system user identifier.
ADBSECCC	blanks	Security file cipher code.
ADBSECDB	0	Security file database identifier.
ADBSECFL	16	Security file number.
ADBSECPW	blanks	Security file password.
ADBSYSCC	blanks	System file cipher code.
ADBSYSDB	0	System file database identifier.
ADBSYSFL	15	System file number.
ADBSYSPW	blanks	System file password.

ADBEUSE Systems Options Default Values

The systems options shown in Table A1.1 on page 116 set default values to be used by the interface view engine to control various run-time characteristics. You can override some of these settings by specifying data set options in a SAS procedure.

Table A1.1 ADBEUSE Systems Options Default Values

Option	Default	Purpose
ADBBYMD	R	BY key mode processing: R - generate return code when adding a new periodic group to a record that has reached the maximum number of periodic group occurrences. N - add a new record.

Option	Default	Purpose
ADBDBMD	M	Database execution mode: M - multi-user S - single user
ADBDEFW	0	Default WHERE clause setting: 0 - creates a default WHERE clause for ADABAS data field formats as follows (assuming that the data field named AA is the first ADABAS descriptor file): where (aa<=0) or aa>0 (numeric) where aa>= ''(alphanumeric blank) 1 - creates a default WHERE clause for ADABAS data field formats as follows (assuming that the data field named AA is the first ADABAS descriptor file): where aa=0 (numeric) where aa= ''(alphanumeric blank) 2 - displays an error return code. A view WHERE clause or SAS WHERE clause is required when random access is desired or when performing updates.
ADBDEL	N	Deleting periodic group flag: N - nulls periodic group values when (1) more than one occurrence still exists or (2) other periodic groups exist within the ADABAS file but are not represented in the view descriptor. P - always deletes the record, regardless of the existence of periodic group fields. The P means "to prune." When ADBDEL=P, you want to remove (reduce) what is superfluous, in this case, the entire logical record.
ADBDELIM	\	View WHERE clause delimiter.
ADBUPD	U	Engine authorization code: U - authorized to perform updates. R - read authorization only.
ADBFMTL	500	ADABAS format buffer length. Minimum value = 100.
ADBISNL	5000	ADABAS ISN buffer length. Minimum value = 100.
ADBMAXM	191	Maximum multiple-value occurrence number.
ADBMAXP	9	Maximum periodic group occurrence number.
ADBMINM	1	Minimum multiple-value occurrence number.
ADBRECL	7500	ADABAS record buffer length. Minimum value = 2100. Maximum value = 32767.

Option	Default	Purpose
ADBSCHL	500	ADABAS search buffer length. Minimum value = 100.
ADBSPANS	*	View WHERE clause SPANS character.
ADBUISN	Y	User ISN flag: Y - user can specify ISN value when adding new records. N - user cannot specify ISN values.
ADBVALL	300	ADABAS value buffer length. Minimum value = 100. Maximum value = 32767.

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

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

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

ISBN 1–58025–546–9

All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, by 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 by the government is subject to restrictions as 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.