



SAS Publishing



SAS/ACCESS[®] 9.1

Interface to ADABAS

Reference

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2004. *SAS/ACCESS® 9.1 Interface to ADABAS: Reference*. Cary, NC: SAS Institute Inc.

SAS/ACCESS® 9.1 Interface to ADABAS: Reference

Copyright © 2004, SAS Institute Inc., Cary, NC, USA

ISBN 1-59047-215-2

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 this 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, January 2004

SAS Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at support.sas.com/pubs or call 1-800-727-3228.

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.

Contents

What's New v
Overview v

PART 1 SAS/ACCESS Interface to ADABAS: Usage 1

Chapter 1 \triangle Overview of the SAS/ACCESS Interface to ADABAS 3

Introduction to the SAS/ACCESS Interface to ADABAS 3
Purpose of the SAS/ACCESS Interface to ADABAS 3
SAS/ACCESS Descriptor Files for ADABAS 4
Example Data in the ADABAS Document 5

Chapter 2 \triangle ADABAS Essentials 7

Introduction to ADABAS Essentials 7
ADABAS DBMS 8
ADABAS Databases 8
ADABAS Data Fields 11
ADABAS Null Values 14
ADABAS and NATURAL Security Options 14

Chapter 3 \triangle ADABAS Data in SAS Programs 17

Introduction to Using ADABAS Data in SAS Programs 17
Reviewing ADABAS Variables 18
Printing ADABAS Data 19
Charting ADABAS Data 20
Calculating Statistics with ADABAS Data 22
Selecting and Combining ADABAS Data 24
Updating a SAS Data File with ADABAS Data 32
Performance Considerations 34

Chapter 4 \triangle Browsing and Updating ADABAS Data 37

Introduction to Browsing and Updating ADABAS Data 37
Browsing and Updating ADABAS Data with the SAS/FSP Procedures 38
Browsing and Updating ADABAS Data with the SQL Procedure 43
Browsing Data with the SELECT Statement 43
Updating Data with the UPDATE Statement 45
Inserting and Deleting Data with the INSERT and DELETE Statements 47
Appending ADABAS Data with the APPEND Procedure 49

PART 2 SAS/ACCESS Interface to ADABAS: Reference 53

Chapter 5 \triangle ACCESS Procedure Reference 55

Introduction to ACCESS Procedure Reference 55

Case Sensitivity in the ACCESS Procedure	56
ACCESS Procedure Syntax for ADABAS	56
SAS Passwords for SAS/ACCESS Descriptors	58
Invoking the ACCESS Procedure	60
ACCESS PROCEDURE Statements for ADABAS	61
WHERE Clause in an ADABAS View Descriptor	88
SORT Clause in a View Descriptor	93
Creating and Using ADABAS View Descriptors Efficiently	94
ACCESS Procedure Formats and Informats for ADABAS	95
Effects of the SAS/ACCESS Interface on ADABAS Data	97

PART 3 Appendices 99

Appendix 1 \triangle Information for the Database Administrator	101
Introduction to the Information for the Database Administrator	101
How the SAS/ACCESS Interface to ADABAS Works	102
Effects of Changing an ADABAS File or NATURAL DDM on Descriptor Files	107
Data Security with ADABAS	108
Controlling Data Locks with ADABAS	111
Maximizing ADABAS Performance	111
Debug Information for ADABAS	112
System Options for PROC ACCESS and the Interface View Engine	112
Appendix 2 \triangle Advanced Topics	115
Introduction to Advanced Topics	115
Data Set Options for ADABAS	116
Using Multiple View Descriptors	118
Deleting an ADABAS Observation	118
Adding an ADABAS Observation	118
Using a BY Key To Resolve Ambiguous Inserts	119
Missing Values (Nulls)	121
Using Multiple-Value Fields in Selection Criteria	122
Periodic Group Fields in Selection Criteria	124
Using a SAS WHERE Clause for Selection Criteria	127
Deciding How to Specify Selection Criteria	130
Appendix 3 \triangle Example Data	131
Introduction to the ADABAS Example Data	132
ADABAS Files	132
NATURAL DDMs Based on the ADABAS Files	145
Access Descriptors for ADABAS	147
View Descriptors Based on the Access Descriptors for ADABAS	150
SAS Data Files for ADABAS	152
Appendix 4 \triangle Recommended Reading	155
Recommended Reading	155
Glossary	157
Index	163

What's New

Overview

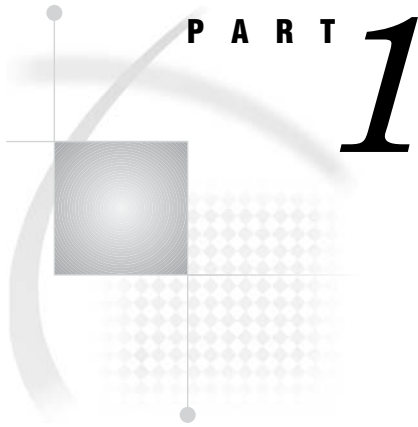
Prior to SAS 9.1, the ADABAS engine could not process a WHERE clause and ISN option. The ADABAS engine read all of the records in the ADABAS table and returned the records to SAS, then applied the WHERE clause.

Now, the ADABAS engine issues an L1 command to the ADABAS table that enables ADABAS to process the WHERE clause and ISN option. With this method, only one ADABAS record is read instead of the complete table, which results in a performance enhancement. See “Retrievals with Only a WHERE Clause” on page 105 for more information.

Note:

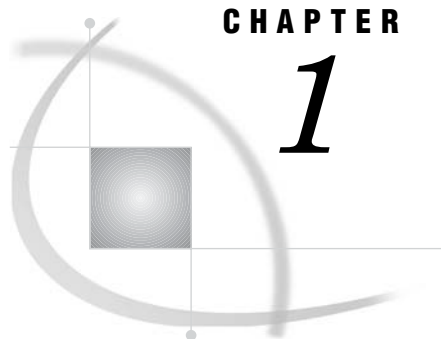
- This section describes the only feature of the SAS/ACCESS Interface to ADABAS that has changed since SAS 8.2.
- z/OS is the successor to the OS/390 operating system. SAS/ACCESS 9.1 for ADABAS is supported on both OS/390 and z/OS operating systems and, throughout this document, any reference to z/OS also applies to OS/390, unless otherwise stated.

△



SAS/ACCESS Interface to ADABAS: Usage

<i>Chapter 1</i>	Overview of the SAS/ACCESS Interface to ADABAS	<i>3</i>
<i>Chapter 2</i>	ADABAS Essentials	<i>7</i>
<i>Chapter 3</i>	ADABAS Data in SAS Programs	<i>17</i>
<i>Chapter 4</i>	Browsing and Updating ADABAS Data	<i>37</i>



CHAPTER

1

Overview of the SAS/ACCESS Interface to ADABAS

<i>Introduction to the SAS/ACCESS Interface to ADABAS</i>	3
<i>Purpose of the SAS/ACCESS Interface to ADABAS</i>	3
<i>SAS/ACCESS Descriptor Files for ADABAS</i>	4
<i>Access Descriptor Files</i>	5
<i>View Descriptor Files</i>	5
<i>Example Data in the ADABAS Document</i>	5

Introduction to the SAS/ACCESS Interface to ADABAS

This section introduces you to SAS/ACCESS software and briefly describes how to use the interface. This section also introduces the sample ADABAS data, SAS/ACCESS descriptor files, and SAS data files used in this document.

Purpose of the SAS/ACCESS Interface to ADABAS

SAS/ACCESS software provides an interface between SAS and the ADABAS database management system (DBMS). With the SAS/ACCESS interface, you can perform the following tasks:

- create SAS/ACCESS descriptor files using the ACCESS procedure
- directly access ADABAS data from within a SAS program using the SAS/ACCESS descriptor files created with the ACCESS procedure
- extract ADABAS data and place it in a SAS data file using the ACCESS procedure, the DATA step, or other SAS procedures
- update ADABAS data using the SQL procedure, SAS/FSP software, SAS/AF software, and the APPEND procedure.

The SAS/ACCESS interface consists of two parts:

- the ACCESS procedure, which you use to define the SAS/ACCESS descriptor files
- the interface view engine, which enables you to use ADABAS data in SAS programs in much the same way as you use SAS data files.

The ACCESS procedure enables you to describe ADABAS data to SAS. You store the description in SAS/ACCESS descriptor files, which you can use in SAS programs much as you would use SAS data files. You can print, plot, and chart the data described by the descriptor files, use it to create other SAS data sets, and so on. Several examples of using ADABAS data in SAS programs are presented in Chapter 3, “ADABAS Data in SAS Programs,” on page 17. Using SAS/ACCESS descriptor files to update ADABAS

data from within a SAS program is shown in Chapter 4, “Browsing and Updating ADABAS Data,” on page 37.

The interface view engine is an integral part of the SAS/ACCESS interface, but the interface’s design is transparent, so you seldom have to deal directly with the engine. SAS automatically interacts with the engine (via the SAS/ACCESS descriptor files) when you use ADABAS data in your SAS programs. SAS and the interface view engine do much of the work automatically, so you can simply use ADABAS data in SAS programs in much the same way you use SAS data.

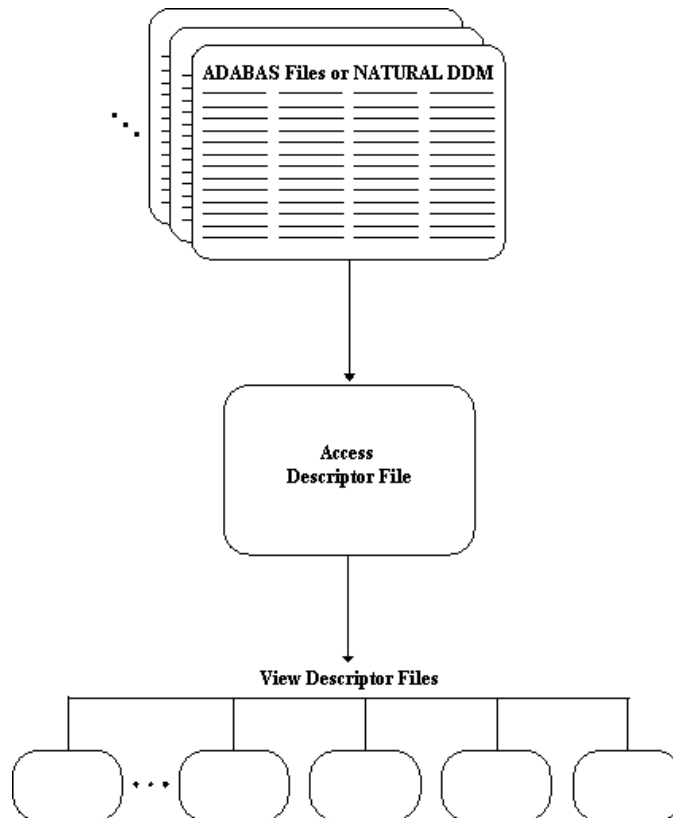
SAS/ACCESS Descriptor Files for ADABAS

SAS/ACCESS software uses SAS/ACCESS descriptor files to establish a connection between SAS and ADABAS. You create these files with the ACCESS procedure.

There are two types of SAS/ACCESS descriptor files: *access descriptors* and *view descriptors*.

The following figure illustrates the relationship among ADABAS data, an access descriptor, and view descriptors.

Figure 1.1 Relationship among ADABAS Data, an Access Descriptor, and View Descriptors



Access Descriptor Files

Access descriptor files are of member type ACCESS. Each access descriptor holds essential information about the ADABAS data you want to access, for example, the ADABAS file number or NATURAL Data Definition Module (DDM) name, the data field names, and their data types. It also contains corresponding information related to SAS, such as the SAS variable names, formats, and informats.

An access descriptor can describe only one ADABAS file or DDM; that is, you cannot join two ADABAS files or DDMs with a single access descriptor.

View Descriptor Files

View descriptor files are sometimes called *views* because their member type is VIEW. This document uses the term *view descriptor* to distinguish them from views that are created by the SAS SQL procedure.

Each view descriptor can define all of the data or a particular subset of the data described by one access descriptor (and therefore one ADABAS file or DDM). For example, you might want to use only three or four possible data fields and only some of the logical records. The view descriptor enables you to select the data fields you want and, by specifying selection criteria, to select only the specific data you want. For example, your selection criteria might be that the date of transaction is July 3, 1998, and that customers' names begin with W.

Typically, for each access descriptor, you will have several view descriptors, selecting different subsets of data.

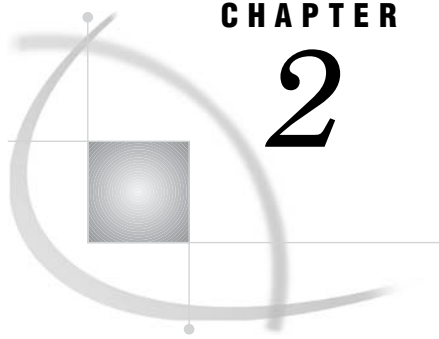
You can join data from multiple ADABAS files or NATURAL DDMs with SAS SQL procedure. The SQL procedure can join data from SAS data files, PROC SQL views, and SAS/ACCESS view descriptors into one resulting file. In addition, SAS/ACCESS view descriptors can come from different database management systems. For examples that use the SQL procedure, see Chapter 3, "ADABAS Data in SAS Programs," on page 17 and Chapter 4, "Browsing and Updating ADABAS Data," on page 37.

Example Data in the ADABAS Document

This document uses several NATURAL DDMs to show you how to use the SAS/ACCESS interface to ADABAS. The data was created for an international textile manufacturer. This company's product line includes some special fabrics that are made to precise specifications. The DDMs are named CUSTOMERS, EMPLOYEE, INVOICE, and ORDER. All the data is fictitious.

The ADABAS data is designed to show how the interface treats ADABAS data. It is not meant as an example for you to follow in designing ADABAS files or NATURAL DDMs for any purpose.

Appendix 3, "Example Data," on page 131 gives more information about the ADABAS data, SAS/ACCESS descriptor files, and SAS data files used in examples. The information about the ADABAS data includes the ADABAS statements that created each file, the data each ADABAS file contains, and a description of the NATURAL DDMs. The information about the SAS/ACCESS descriptor files includes their definitions and any selection criteria that were specified for them. The information about the SAS data files includes the SAS statements that created each data file and the data that each contains.



CHAPTER

2

ADABAS Essentials

<i>Introduction to ADABAS Essentials</i>	7
<i>ADABAS DBMS</i>	8
<i>ADABAS Databases</i>	8
<i>ADABAS Files</i>	9
<i>ADABAS File Number</i>	9
<i>Level Number</i>	9
<i>Data Field Names</i>	9
<i>Logical Record ISN</i>	9
<i>NATURAL Data Definition Modules</i>	10
<i>DDM File Name</i>	10
<i>Data Field Names</i>	10
<i>ADABAS Descriptors</i>	10
<i>Subdescriptor</i>	10
<i>Superdescriptor</i>	10
<i>Phonetic Descriptor</i>	11
<i>ADABAS Data Fields</i>	11
<i>Data Field Types</i>	11
<i>Elementary Field</i>	11
<i>Multiple-value Field</i>	11
<i>Group Field</i>	11
<i>Periodic Group Field</i>	12
<i>Subfield</i>	12
<i>Superfield</i>	12
<i>Mapping Data between SAS and ADABAS</i>	12
<i>Data Field Formats and Lengths</i>	13
<i>ADABAS Null Values</i>	14
<i>ADABAS and NATURAL Security Options</i>	14
<i>ADABAS Security Options</i>	14
<i>NATURAL Security Options</i>	15

Introduction to ADABAS Essentials

This section introduces SAS users to ADABAS, Software AG's database management system (DBMS). The section focuses on the following terms and concepts

- the ADABAS DBMS and ADABAS databases
- ADABAS files, NATURAL Data Definition Modules, and ADABAS descriptors (indexes)
- ADABAS data fields and ADABAS and NATURAL data formats and lengths
- null (missing) values

- ADABAS Security and NATURAL SECURITY System options.

If you want more information about an ADABAS concept or term than this section provides, see the ADABAS information about your system.

ADABAS DBMS

ADABAS is Software AG's database management system (DBMS). ADABAS organizes and accesses data according to relationships among *data fields*. The relationships among data fields are expressed by *ADABAS files*, which consist of *data fields* and *logical records*.

With the ADABAS DBMS, you can also use the high-level language NATURAL to operate on data that is managed by the DBMS. NATURAL is Software AG's fourth generation application development system that enables you to create, modify, read, and protect data that the DBMS manages. All ADABAS files and data fields referenced in a NATURAL program must be defined to NATURAL through a Data Definition Module (DDM).

ADABAS has single-user and multi-user execution environments, both of which are supported by the SAS/ACCESS interface to ADABAS.

ADABAS Databases

An ADABAS database is a collection of data organized into ADABAS files. Each database has an associated *database identifier*, which is a numerical value in the range 1 to 65,535, and a *database name*, which is a character value with a maximum of 16 characters. Each database can consist of up to 5,000 logical files.

An ADABAS database consists of three system files: Data Storage, Associator, and Work Storage.

- The Data Storage system file contains the actual data records for all ADABAS files in a database, in compressed form.
- The Associator system file contains internal storage information that manages the data for the entire database.
- The Work Storage system file contains temporary work files.

To use the SAS/ACCESS interface to ADABAS, you need to be familiar with three ADABAS components: ADABAS files, NATURAL DDMs, and ADABAS descriptors (which is an ADABAS data field that provides an index of its values). ADABAS files and NATURAL DDMs are the components from which you create SAS/ACCESS access descriptor and view descriptor files. Knowing about ADABAS descriptors can help you minimize ADABAS's processing time for your SAS/ACCESS view descriptors.

Note: To avoid confusion, keep in mind the two usages of the term *descriptor* throughout this document:

- An *ADABAS descriptor* is an ADABAS data field that provides an index of the data field's values.
- *SAS/ACCESS descriptor files*, on the other hand, are the files used to establish a connection between SAS and ADABAS.

△

The following sections describe ADABAS files, NATURAL DDMs, and ADABAS descriptors.

ADABAS Files

An ADABAS file is a collection of logically related data, organized by data fields and logical records. ADABAS permits maximums of 926 data fields and 4,294,967,294 logical records in each ADABAS file.

The following output illustrates four data fields and seven logical records from an ADABAS file containing data about customers. The data fields are the vertical columns of data. The logical records are the horizontal rows of data.

Output 2.1 Sample ADABAS File

CU	CI	ST	CO
14324742	San Jose	CA	USA
14569877	Memphis	TN	USA
14898029	Rockville	MD	USA
24589689	Belgrade		Yugoslavia
26422096	La Rochelle		France
38763919	Buenos Aires		Argentina
46783280	Singapore		Singapore

ADABAS files are created with the ADABAS utility ADACMP. (To see the ADABAS data definition statements that created the ADABAS files used in this document, refer to Appendix 3, “Example Data,” on page 131.)

ADABAS File Number

When you create an ADABAS file, you assign a file number using the FILE= statement of the ADACMP utility. Each database can consist of up to 5,000 logical files, depending on the device type.

Level Number

A data field *level number* is a one- or two-digit number, from 01 to 07, used in conjunction with data field grouping. (Grouping is discussed in “ADABAS Data Fields” on page 11.) Data fields with a level of 2 or greater are considered to be a part of the immediately preceding group, which has a lower level number.

Data Field Names

ADABAS data fields are identified by a two-character name. Each data field name in an ADABAS file must be unique. The first character must be alphabetic, and the second character can be either alphabetic or numeric. For example, AA and B4 are valid data field names.

Logical Record ISN

Each logical record within an ADABAS file is assigned an *internal sequence number* (ISN). An ISN is the logical identifier for each record. ISNs are unique within each ADABAS file.

Note: When you create SAS/ACCESS descriptor files for ADABAS data, the ACCESS procedure creates a SAS variable named ISN. This variable gives you access to the ISNs for all logical records stored in the ADABAS file. △

NATURAL Data Definition Modules

To reference an ADABAS file and its data fields in NATURAL programs, you must create a NATURAL Data Definition Module (DDM) based on the ADABAS file. (Note that a DDM is often referred to as an ADABAS file, even though it is really only a *view* of an actual ADABAS file.) A DDM has an assigned name, which references the ADABAS file number on which the DDM is based. Also, more descriptive data field names can be assigned to a DDM. DDMs are stored in a system file, which is simply another ADABAS file.

DDM File Name

The filename for a NATURAL DDM can be a maximum of 32 characters.

Data Field Names

In a NATURAL DDM, data fields can be assigned a DDM external name of 3 to 32 characters. For example, in the CUSTOMERS DDM, the DDM data field name CUSTOMER corresponds to the ADABAS file two-character data field name CU.

ADABAS Descriptors

If you plan to use a data field often in selection criteria, you can designate it as a key field. You designate a key field by specifying the descriptor option in the ADACMP utility data definition statement. When a data field is a descriptor field, ADABAS maintains and stores its values in an *inverted list*. An inverted list contains the different values of a descriptor data field, along with the count and the ISNs of the logical records that contain each value. ADABAS descriptors can also be defined so that inverted lists contain unique values only.

Specifying ADABAS descriptors speeds up the selection process considerably since ADABAS is able to access key values directly. Also, specifying descriptors controls read sequence when reading ADABAS data in sequential order.

Several descriptor types can be specified for a data field. Each descriptor type is explained below.

Note: In order for you to use SAS variables corresponding to ADABAS data fields in a SAS BY statement, an SQL ORDER BY clause, or a view SORT clause, the data field must be designated as an ADABAS descriptor. Regarding a WHERE clause, there are conditions when you can use a nondescriptor data field and when you must use a descriptor data field. These conditions are explained in Chapter 5, “ACCESS Procedure Reference,” on page 55. △

Subdescriptor

A subdescriptor is an ADABAS descriptor that is derived from a portion of an elementary data field. For example, if ZIPCODE is a data field, a subdescriptor for it could be ZIPLAST2 defined for the last two digits of a zipcode.

You can include a subdescriptor in SAS/ACCESS descriptor files for retrieval and selection criteria, but you cannot use subdescriptors in SAS updating procedures.

Superdescriptor

A superdescriptor is an ADABAS descriptor derived from more than one data field, portions of data fields, or combinations thereof. For example, a superdescriptor named

STATE-ZIPLAST2 could be defined for the first two digits from the STATE data field and the last two digits from the ZIPCODE data field.

You can include a superdescriptor in SAS/ACCESS descriptor files for retrieval and selection criteria, but you cannot use superdescriptors in SAS updating procedures.

Phonetic Descriptor

A phonetic descriptor is an ADABAS descriptor defined to perform searches based on phonetic values, for example, retrieval by family name.

You can include a phonetic descriptor in SAS/ACCESS descriptor files for retrieval and selection criteria, but you cannot use phonetic descriptors in SAS updating procedures.

Note that if you use a phonetic descriptor in a SAS WHERE clause, the interface view engine must be able to process the entire SAS WHERE clause.

Note: The hyperdescriptor type is not described because hyperdescriptors are not supported by the SAS/ACCESS interface to ADABAS. Your ADABAS file can contain hyperdescriptors, but they will be ignored. △

ADABAS Data Fields

You can group logically related ADABAS data fields into one ADABAS file, which consequently can be accessed by one NATURAL DDM. Up to 926 data fields can be contained in a single logical record. Data fields have assigned types, formats, and lengths.

Data Field Types

The SAS/ACCESS interface to ADABAS supports the ADABAS data fields as described below.

Elementary Field

An elementary field is limited to one value per record. For example, LASTNAME could be an elementary field.

Multiple-value Field

A multiple-value field can have 0 to 191 values per record. For example, JOBTITLE could be a multiple-value field because each employee at a company could have multiple job titles during his or her employment.

Group Field

A group field is several consecutive data fields combined into one for efficient access and ease of reference. Defining a group field enables you to reference a series of data fields by using a group name. For example, a group field named EDUCATION could consist of these data fields: COLLEGE, DEGREE, and YEAR.

A group field can also consist of other groups. In conjunction with grouping, you can assign level numbers 01 to 07 to define a group.

Periodic Group Field

A periodic group field is a group of data fields that repeat. A periodic group can be repeated up to 191 times and can contain one or more elementary fields and multiple-value fields. Groups can be nested, but periodic groups cannot. One periodic group cannot contain another. However, a record can have several different periodic groups.

Subfield

A subfield is a data field defined from a portion of another data field. For example, a subfield named AREA-CODE could be defined for the first three digits from the PHONE data field.

You use subfields for read operations only; they cannot be used for updating directly.

Superfield

A superfield is a data field composed of several data fields, portions of fields, or combinations thereof. For example, a superfield could be STATE-AREA-CODE accessing such values as TX512, NM505, and CA213.

You use superfields for read operations only; they cannot be used for updating directly.

Mapping Data between SAS and ADABAS

When you access ADABAS data through the SAS/ACCESS interface, the interface view engine maps the ADABAS data into SAS observations. You need to be aware of how the interface view engine maps multiple-value fields and periodic groups. That is, *multiple-value field occurrences are mapped to multiple SAS variables, and periodic group occurrences are mapped to multiple SAS observations.*

For example, suppose an ADABAS file has the data fields and values shown in the following output. LASTNAME is an elementary field, JOBTITLE is a multiple-value field, and EDUCATION is a periodic group consisting of the data fields COLLEGE, DEGREE, and YEAR.

Output 2.2 ADABAS Data

LASTNAME	JOBTITLE	EDUCATION		
Reid	Systems Analyst	Purdue	BA	1973
	DBA	Harvard	MBA	1975

The interface view engine would map the ADABAS data into two SAS observations, as shown in the following output.

Output 2.3 ADABAS Data Mapped into SAS Observations

LASTNAME	JOBTITL1	JOBTITL2	COLLEGE	DEGREE	YEAR
Reid	Systems Analyst	DBA	Purdue	BA	1973
Reid	Systems Analyst	DBA	Harvard	MBA	1975

If you were browsing the ADABAS data, such as with the FSVIEW procedure, the results would be similar to Output 2.3, with LASTNAME, JOBTITL1, and JOBTITL2 repeated for each set of COLLEGE, DEGREE, and YEAR values. Actually though, the value Reid is stored in the ADABAS file only once. For retrievals, the results are straightforward. When updating, however, you need to keep in mind how the interface view engine maps multiple-value fields and periodic groups.

Suppose you want to change the spelling of a last name using the FSVIEW procedure. To change Reid to Reed, all you need to do is type REED over one of the REID values, and, with a single update operation, the last names are all changed. On the other hand, suppose you want to delete an observation for Reid using the FSEDIT procedure. Each observation for his job titles and education data would display his last name. If you deleted an observation, for example, the one for Purdue, the deletion would not affect the last name or the job title data, but the Purdue observation would be gone. For more information and an example of deleting an observation from ADABAS data, see Chapter 4, “Browsing and Updating ADABAS Data,” on page 37.

Data Field Formats and Lengths

Data definition statements enable you to define data field formats and lengths for both ADABAS files and NATURAL DDMs. The standard format of a data field is specified with a one-character code shown next in the following table. The standard length of a data field is specified in bytes; the maximum length is also given.

Table 2.1 ADABAS Standard Data Field Formats and Lengths

Data Type	Standard Format	Standard Length	Description
Alphanumeric	A (ADABAS) A (DDM)	253 byte maximum	Left-justified, with trailing blanks removed
Binary	B (ADABAS) B (DDM)	126 byte maximum	Right-justified, unsigned, with leading zeros removed
Fixed Point	F (ADABAS) B (DDM)	Must be 4 bytes	Right-justified, signed, with twos complement notation
Floating Point	G (ADABAS) F (DDM)	Must be 4 or 8 bytes	In normalized form and signed
Packed Decimal	P (ADABAS) P (DDM)	15 byte maximum	Right-justified and signed
Unpacked Decimal (Zoned)	U (ADABAS) N (DDM)	29 byte maximum	Right-justified and signed

If the standard length of a data field is specified as zero, the data field is a *variable length field*, which has no maximum or required length.

Note that when creating SAS/ACCESS descriptor files, you can specify SAS formats for ADABAS data to change the way the data appears. For example, you can add decimal points. Also, you can specify a SAS date format in your SAS/ACCESS descriptor files to designate a date representation.

ADABAS Null Values

ADABAS has a special value called a *null* value, which means an absence of information. A null value is analogous to a missing value in SAS.

You can define data fields not to store null data by specifying the NU option in data definition statements. In normal data storage (that is, NU not specified), a null value is represented by two bytes (one for the value length and one for the null value). Suppressing null values results in a null value being represented by a one-byte empty field indicator. The null value itself is not stored.

Knowing whether a data field allows null values assists you in writing selection criteria and in entering values to update ADABAS data. For example, if the NU option is specified for an ADABAS descriptor data field, null values for the data field are not stored in the inverted list. Therefore, a search using this data field and a null value as the search value, would result in no records selected, even though there might be records that contain a null value for the data field.

For more information about null values, see “Missing Values (Nulls)” on page 121.

ADABAS and NATURAL Security Options

The ADABAS DBMS offers security options through both ADABAS and NATURAL. To protect your ADABAS data, you can use either form of security, or you can have both work together.

ADABAS Security Options

ADABAS provides a security facility to prevent unauthorized access to data stored in ADABAS files. Security is available through password protection and by maintaining data in enciphered form.

passwords provide protection at the ADABAS file level, data field level, and data value level. These security options are defined with the SECURITY utility ADASCR and are stored in the ADABAS Security system file.

To access an ADABAS file protected by a password, you must provide the valid password. Each data field in an ADABAS file can be assigned up to fifteen levels of read and update security. A user password specifies the authority for the data field, and ADABAS automatically determines whether the user is authorized to perform the requested operation. If the permission level of a user's password is equal to or greater than the permission level for the file the user is trying to access, access is granted. Any ADABAS file can be protected on individual data field values. In this case, the password specifies value restrictions on logical records to be selected, read, and updated.

cipher codes are simple numeric codes that you can assign using the ADACMP utility when creating an ADABAS file. Ciphering renders data records unreadable when they are not displayed with an ADABAS program or utility. You must supply this cipher code in order to access the enciphered data.

Note: System information such as DDM and NATURAL SECURITY information is also stored in ADABAS files; they too can be password-protected or enciphered. △

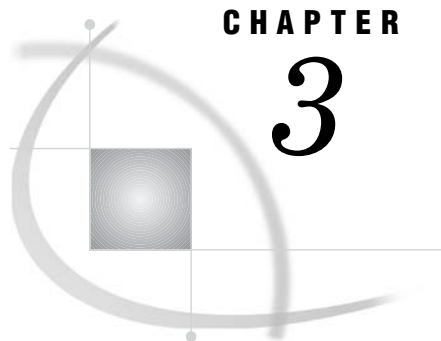
NATURAL Security Options

NATURAL provides an optional security system that controls the access and use of the NATURAL environment. You can restrict the use of whole application systems, individual programs and functions, and the access to DDMs.

Security is accomplished by defining objects and the relationships among these objects. There are three objects that you need to be familiar with when accessing data through NATURAL DDMs with the SAS/ACCESS interface: users, libraries, and files.

- | | |
|-----------|---|
| users | can be people, terminals, or groups of either, with assigned identifiers. The user identifier identifies the user to NATURAL SECURITY and controls user activity during a NATURAL session. The identifier is unique to NATURAL and can be up to eight characters long. Each user identifier can have an associated eight-character password. |
| libraries | contain sets of NATURAL source programs and/or object modules that perform a particular function, with assigned identifiers. Stored in the library data are the ADABAS passwords or cipher codes to enable NATURAL programs to work with ADABAS Security. The library identifier identifies the library and the ADABAS file it is authorized to access to NATURAL SECURITY. The identifier is unique to NATURAL and can be up to eight characters long. |
| files | are the NATURAL DDMs based on ADABAS files. |

Relationships, called *Links*, are defined among these objects. These links define which users are allowed to use a library and which files a library is allowed to access. The users, libraries, files, and links are all stored in the NATURAL Security system file, which can also be protected with an ADABAS password or cipher code since it is an ADABAS file. For example, one user identifier and library might be able to access a DDM for read only, while another user identifier and library might be able to read and update the same DDM.



CHAPTER

3

ADABAS Data in SAS Programs

<i>Introduction to Using ADABAS Data in SAS Programs</i>	17
<i>Reviewing ADABAS Variables</i>	18
<i>Printing ADABAS Data</i>	19
<i>Charting ADABAS Data</i>	20
<i>Calculating Statistics with ADABAS Data</i>	22
<i>Calculating Statistics Using the FREQ Procedure</i>	22
<i>Calculating Statistics Using the MEANS Procedure</i>	22
<i>Calculating Statistics Using the RANK Procedure</i>	24
<i>Selecting and Combining ADABAS Data</i>	24
<i>Selecting and Combining Data Using the WHERE Statement</i>	25
<i>Selecting and Combining Data Using the SQL Procedure</i>	26
<i>Combining Data from Various Sources</i>	26
<i>Creating New Variables with the GROUP BY Clause</i>	31
<i>Updating a SAS Data File with ADABAS Data</i>	32
<i>Performance Considerations</i>	34

Introduction to Using ADABAS Data in SAS Programs

An advantage of the SAS/ACCESS interface to ADABAS is that it enables SAS to read and write ADABAS data directly using SAS programs. This section presents examples using ADABAS data accessed through view descriptors as input data for SAS programs.

Throughout the examples, the SAS terms *variable* and *observation* are used instead of comparable ADABAS terms because this section illustrates how to use SAS procedures and the DATA step. The examples demonstrate how to print and chart data, how to use the SQL procedure to combine data from various sources, and how to update a Version 6 SAS data file with ADABAS data. For more information about the SAS language and procedures used in the examples, refer to the documents listed at the end of each section.

“Performance Considerations” on page 34, presents some techniques for using view descriptors efficiently in SAS programs.

For definitions of all view descriptors referenced in this section, see Appendix 3, “Example Data,” on page 131. This appendix also contains the ADABAS data and SAS data files used in this document.

Reviewing ADABAS Variables

If you want to use ADABAS data that is described by a view descriptor in your SAS program but cannot remember the variable names or formats and informats, you can use the CONTENTS or DATASETS procedures to display this information.

The following examples use the DATASETS procedure to give you information about the view descriptor VLIB.CUSPHON, which references the NATURAL DDM named CUSTOMERS.

```
proc datasets library=vlib memtype=view;
  contents data=cusphon;
quit;
```

The following output shows the information for this example. The data that is described by VLIB.CUSPHON is shown in Output 3.9.

Output 3.1 Results of Using the DATASETS Procedure to Review a View Descriptor

The SAS System							
DATASETS PROCEDURE							
Data Set Name:	VLIB.CUSPHON			Observations:	.		
Member Type:	VIEW			Variables:	3		
Engine:	SASIOADB			Indexes:	0		
Created:	14:09 Friday, October 5, 1990			Observation Length:	80		
Last Modified:	14:33 Friday, October 5, 1990			Deleted Observations:	0		
Data Set Type:				Compressed:	NO		
Label:							
-----Engine/Host Dependent Information-----							
-----Alphabetic List of Variables and Attributes-----							
#	Variable	Type	Len	Pos	Format	Informat	Label
1	CUSTNUM	Char	8	0	\$8.	\$8.	CUSTOMER
3	NAME	Char	60	20	\$60.	\$60.	NAME
2	PHONE	Char	12	8	\$12.	\$12.	TELEPHONE

Note the following points about this output:

- You cannot change a view descriptor's variable labels using the DATASETS procedure. The labels are generated to be the complete ADABAS data field name when the view descriptor is created and therefore cannot be overwritten.
- The **Created** date is the date the access descriptor for this view descriptor was created.
- The **Last Modified** date is the last time the view descriptor was updated.
- The **Observations** number field contains a null value.

For more information about the DATASETS procedure, see the *Base SAS Procedures Guide*.

Printing ADABAS Data

Printing ADABAS data that is described by a view descriptor is like printing any other SAS data set, as shown in the following examples.

The following example contains the code for printing the ADABAS data that is described by the view descriptor VLIB.EMPINFO:

```
proc print data=vlib.empinfo;
  title "Brief Employee Information";
run;
```

VLIB.EMPINFO accesses data from the NATURAL DDM named EMPLOYEE. The following output shows the results for this example.

Output 3.2 Results of Printing ADABAS Data

Brief Employee Information				
OBS	EMPID	DEPT	LASTNAME	
1	119012	CSR010	WOLF-PROVENZA	
2	120591	SHP002	HAMMERSTEIN	
3	123456		VARGAS	
4	127845	ACC024	MEDER	
5	129540	SHP002	CHOULAI	
6	135673	ACC013	HEMESLY	
7	212916	CSR010	WACHBERGER	
8	216382	SHP013	PURINTON	
9	234967	CSR004	SMITH	
10	237642	SHP013	BATTERSBY	
11	239185	ACC024	DOS REMEDIOS	
12	254896	CSR011	TAYLOR-HUNYADI	
13	321783	CSR011	GONZALES	
14	328140	ACC043	MEDINA-SIDONIA	
15	346917	SHP013	SHIEKELESLAM	
16	356134	ACC013	DUNNETT	
17	423286	ACC024	MIFUNE	
18	456910	CSR010	ARDIS	
19	456921	SHP002	KRAUSE	
20	457232	ACC013	LOVELL	
21	459287	SHP024	RODRIGUES	
22	677890	CSR010	NISHIMATSU-LYNCH	

When you use the PRINT procedure, you might want to use the OBS= option, which enables you to specify the last observation to be processed. This is especially useful when the view descriptor describes large amounts of data or when you just want to see an example of the output. The following example uses the OBS= option to print the first five observations described by the view descriptor VLIB.CUSORDR.

```
proc print data=vlib.cusordr (obs=5);
  title "First Five Observations Described
        by VLIB.CUSORDR";
run;
```

VLIB.CUSORDR accesses data from the NATURAL DDM named ORDER. The following output shows the result of this example.

Output 3.3 Results of Using the OBS= Option

First Five Observations Described by VLIB.CUSORDR		
OBS	STOCKNUM	SHIPTO
1	9870	19876078
2	1279	39045213
3	8934	18543489
4	3478	29834248
5	2567	19783482

In addition to the OBS= option, the FIRSTOBS= option also works with view descriptors. The FIRSTOBS= option does not improve performance significantly because each observation must still be read and its position calculated. The POINT= option in the SET statement is not currently supported by the SAS/ACCESS interface to ADABAS.

For more information about the PRINT procedure, see the *Base SAS Procedures Guide*. For more information about the OBS= and FIRSTOBS= options, see the *SAS Language Reference: Dictionary*.

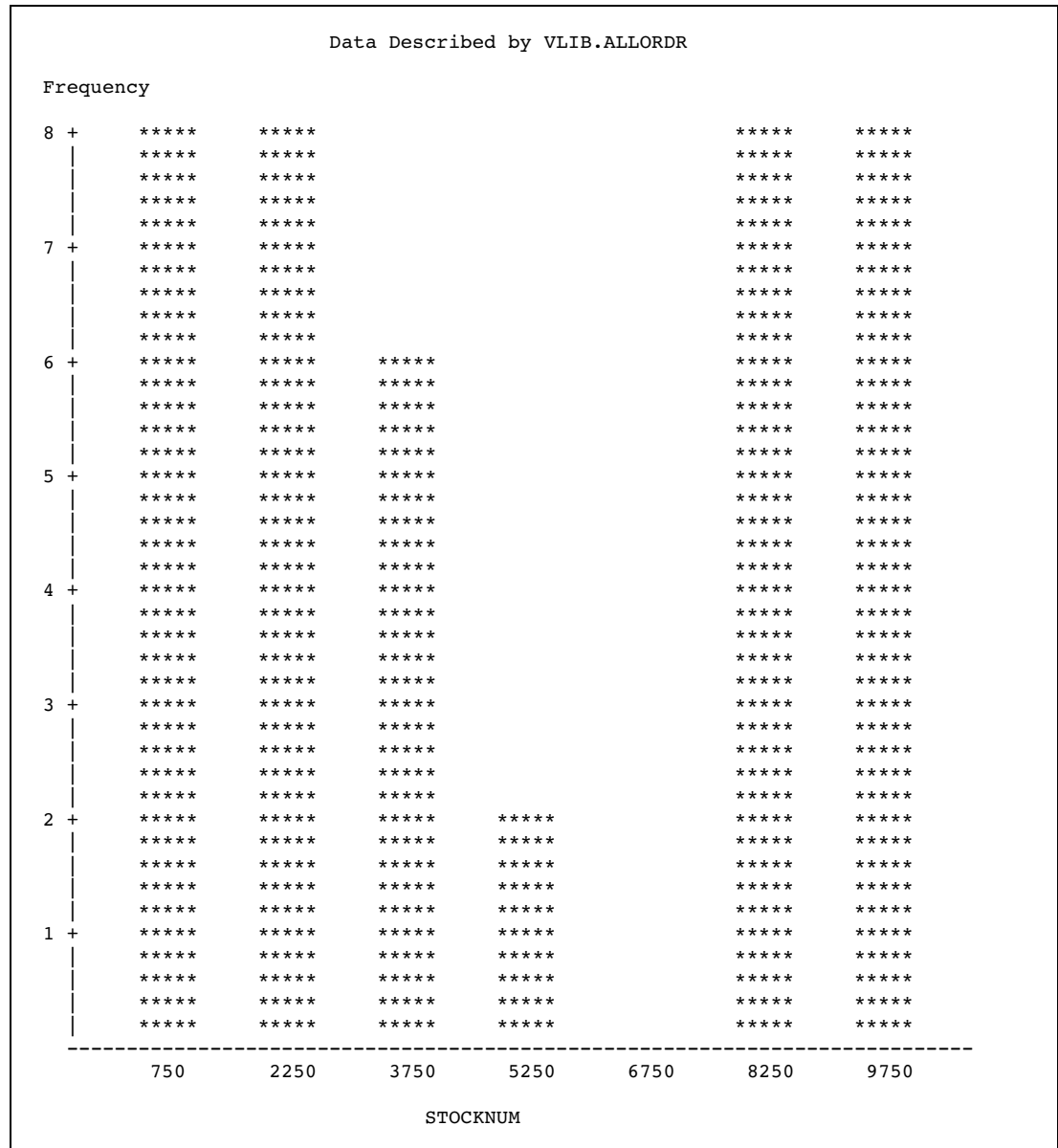
Charting ADABAS Data

CHART procedure programs work with ADABAS data that is described by view descriptors just as they do with other SAS data sets. The following example uses the view descriptor VLIB.ALLORDR to create a vertical bar chart of the number of orders per product.

```
proc chart data=vlib.allordr;
  vbar stocknum;
  title "Data Described by VLIB.ALLORDR";
run;
```

VLIB.ALLORDR accesses data from the NATURAL DDM named ORDER. The following output shows the results for this example. STOCKNUM represents each product; the number of orders for each product is represented by the height of the bar.

Output 3.4 Results of Charting ADABAS Data



For more information about the CHART procedure, see the *Base SAS Procedures Guide*.

If you have SAS/GRAPH software, you can create colored block charts, plots, and other graphics based on ADABAS data. See the *SAS/GRAPH Reference, Volumes 1 and 2* for more information about the types of graphics you can produce with SAS/GRAPH software.

Calculating Statistics with ADABAS Data

You can use statistical procedures on ADABAS data that is described by view descriptors just as you would with SAS data files. This section shows simple examples using the FREQ, MEANS, and RANK procedures.

Calculating Statistics Using the FREQ Procedure

Suppose you wanted to find what percentage of your invoices went to each country so that you can decide where to increase your overseas marketing. The following example calculates the percentages of invoices for each country accessed by the NATURAL DDM named INVOICE, using the view descriptor VLIB.INV.

```
proc freq data=vlib.inv;
  tables country;
  title "Data Described by VLIB.INV";
run;
```

The following output shows the one-way frequency table this example generates.

Output 3.5 Results of Calculating Statistics Using the FREQ Procedure

Data Described by VLIB.INV				
COUNTRY				
COUNTRY	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Argentina	2	11.8	2	11.8
Australia	1	5.9	3	17.6
Brazil	4	23.5	7	41.2
USA	10	58.8	17	100.0
Frequency Missing = 2				

For more information about the FREQ procedure, see the *Base SAS Procedures Guide*.

Calculating Statistics Using the MEANS Procedure

In an analysis of recent orders, suppose you also want to determine some statistics for each of your USA customers. In the following SAS program, the view descriptor VLIB.USAORDR accesses data from the NATURAL DDM named ORDER, the SAS WHERE statement selects observations that have a SHIPTO value beginning with a 1, which indicates a USA customer, and the SAS BY statement sorts the data by order number. (Note that both ORDERNUM and SHIPTO are ADABAS descriptor data fields.)

The following example generates the mean and sum of the length of material ordered and the fabric charges for each USA customer. Also included are the number of observations (N) and the number of missing values (NMISS).

```
proc means data=vlib.usaordr mean sum n nmiss
  maxdec=0;
  where shipto like "1%";
```

```

by ordernum;
var length fabricch;
title "Data Described by VLIB.USAORDR";
run;

```

The following output shows the results for this example.

Output 3.6 Results of Calculating Statistics Using the MEANS Procedure

Data Described by VLIB.USAORDR					
----- ORDERNUM=11269 -----					
Variable	Label	N	Nmiss	Mean	Sum
LENGTH	LENGTH	1	0	690	690
FABRICCH	FABRICCHARGES	1	0	0	0
----- ORDERNUM=11271 -----					
Variable	Label	N	Nmiss	Mean	Sum
LENGTH	LENGTH	1	0	110	110
FABRICCH	FABRICCHARGES	1	0	11063836	11063836
----- ORDERNUM=11273 -----					
Variable	Label	N	Nmiss	Mean	Sum
LENGTH	LENGTH	1	0	450	450
FABRICCH	FABRICCHARGES	1	0	252149	252149
----- ORDERNUM=11274 -----					
Variable	Label	N	Nmiss	Mean	Sum
LENGTH	LENGTH	1	0	1000	1000
FABRICCH	FABRICCHARGES	1	0	0	0
----- ORDERNUM=11276 -----					
Variable	Label	N	Nmiss	Mean	Sum
LENGTH	LENGTH	1	0	1500	1500
FABRICCH	FABRICCHARGES	1	0	1934460	1934460
----- ORDERNUM=11278 -----					
Variable	Label	N	Nmiss	Mean	Sum
LENGTH	LENGTH	1	0	2500	2500
FABRICCH	FABRICCHARGES	1	0	1400825	1400825

For more information about the MEANS procedure, see the *Base SAS Procedures Guide*.

Calculating Statistics Using the RANK Procedure

You can use advanced statistics procedures on ADABAS data that is described by a view descriptor. The following example uses the RANK procedure to calculate the order of birthdays for a set of employees. This example creates a SAS data file MYDATA.RANKEX from the view descriptor VLIB.EMPS and assigns the name DATERANK to the new variable (in the data file) created by the procedure.

```
proc rank data=vlib.emps out=mydata.rankek;
  var birthdat;
  ranks daterank;
run;
proc print data=mydata.rankek;
  title "Order of Employee Birthdays";
run;
```

VLIB.EMPS accesses data from the NATURAL DDM named EMPLOYEE. The following output shows the result of this example.

Output 3.7 Results of Calculating Statistics Using the RANK Procedure

Order of Employee Birthdays					
OBS	EMPID	JOBCODE	BIRTHDAT	LASTNAME	DATERANK
1	456910	602	24SEP53	ARDIS	5
2	237642	602	13MAR54	BATTERSBY	6
3	239185	602	28AUG59	DOS REMEDIOS	7
4	321783	602	03JUN35	GONZALES	2
5	120591	602	12FEB46	HAMMERSTEIN	4
6	135673	602	21MAR61	HEMESLY	8
7	456921	602	12MAY62	KRAUSE	9
8	457232	602	15OCT63	LOVELL	11
9	423286	602	31OCT64	MIFUNE	12
10	216382	602	24JUL63	PURINTON	10
11	234967	602	21DEC67	SMITH	13
12	212916	602	29MAY28	WACHBERGER	1
13	119012	602	05JAN46	WOLF-PROVENZA	3

For more information about the RANK procedure and other advanced statistics procedures, see the *Base SAS Procedures Guide*.

Selecting and Combining ADABAS Data

The great majority of SAS programs select and combine data from various sources. The method you use depends on the configuration of the data. The next three examples show you how to select and combine data using two different methods. When choosing between these methods, you should consider the issues described in “Performance Considerations” on page 34.

Selecting and Combining Data Using the WHERE Statement

Suppose you have two view descriptors, VLIB.USAINV and VLIB.FORINV, that list the invoices for USA and foreign customers, respectively. You can use the SET statement to concatenate these files into a SAS data file containing information about customers who have not paid their bills and whose bills amount to at least \$300,000.

The following example contains the code to create the SAS data file containing the information you want on the customers.

```
data notpaid(keep=invoicecn billedto amtville
  billedon paidon);
  set vlib.usainv vlib.forinv;
  where paidon is missing and
    amtville>=300000;
run;
proc print;
  title "High Bills--Not Paid";
run;
```

In the SAS WHERE statement, you must use the SAS variable names, not the ADABAS data field names. Both VLIB.USAINV and VLIB.FORINV access data in the NATURAL DDM named INVOICE. The following output shows the result of the new temporary data file, WORK.NOTPAID.

Output 3.8 Results of Selecting and Combining Data Using a WHERE statement

High Bills--Not Paid					
OBS	INVOICEN	BILLEDTO	AMTBILLE	BILLEDON	PAIDON
1	12102	18543489	11063836.00	17NOV88	.
2	11286	43459747	12679156.00	10OCT88	.
3	12051	39045213	1340738760.90	02NOV88	.
4	12471	39045213	1340738760.90	27DEC88	.
5	12476	38763919	34891210.20	24DEC88	.

The first line of the DATA step uses the KEEP= data set option. This option works with view descriptors just as it works with other SAS data sets; that is, the KEEP= option specifies that you want only the listed variables to be included in the new data file, NOTPAID, although you can use the other variables within the DATA step.

Notice that the WHERE statement includes two conditions to be met. First, it selects only observations that have missing values for the variable PAIDON. As you can see, it is important to know how the ADABAS data is configured before you can use this data in a SAS program.

Second, the WHERE statement requires that the amount in each bill be higher than a certain figure. Again, you need to be familiar with the ADABAS data so that you can determine a reasonable figure for this expression.

When referencing a view descriptor in a SAS procedure or DATA step, it is more efficient to use a SAS WHERE statement than to use a subsetting IF statement. A DATA step or SAS procedure passes the SAS WHERE statement as a WHERE clause to the interface view engine, which adds it (using the Boolean operator AND) to any WHERE clause defined in the view descriptor. The view descriptor is then passed to

ADABAS for processing. Processing ADABAS data using a WHERE clause might reduce the number of logical records read and therefore often improves performance.

For more information about the SAS WHERE statement, see the *SAS Language Reference: Dictionary*.

Selecting and Combining Data Using the SQL Procedure

This section provides two examples of using the SAS SQL procedure on ADABAS data. The SQL procedure implements the Structured Query Language (SQL) and is included in Base SAS software. The first example illustrates using the SQL procedure to combine data from three sources. The second example shows how to use the PROC SQL GROUP BY clause to create new variables from data that is described by a view descriptor.

Combining Data from Various Sources

Suppose you have the view descriptors VLIB.CUSPHON and VLIB.CUSORDR based on the NATURAL DDMs CUSTOMERS and ORDER, respectively, and a SAS data file, MYDATA.OUTOFSTK, that contains names and numbers of products that are out of stock. You can use the SQL procedure to join all these sources of data to form a single output file. The SAS WHERE or subsetting IF statements would not be appropriate in this case because you want to compare variables from several sources, rather than simply merge or concatenate the data.

The following example contains the code to print the view descriptors and the SAS data file:

```
proc print data=vlib.cusphon;
    title "Data Described by VLIB.CUSPHON";
run;

proc print data=vlib.cusordr;
    title "Data Described by VLIB.CUSORDR";
run;

proc print data=mydata.outofstk;
    title "SAS Data File MYDATA.OUTOFSTK";
run;
```

The following three outputs show the results of the PRINT procedure performed on the data that is described by the view descriptors VLIB.CUSPHON and VLIB.CUSORDER and on the SAS data file MYDATA.OUTOFSTK.

Output 3.9 Data That is Described by the View Descriptor VLIB.CUSPHON

Data Described by VLIB.CUSPHON		
OBS	CUSTNUM	PHONE
1	12345678	919/489-5682
2	14324742	408/629-0589
3	14569877	919/489-6792
4	14898029	301/760-2541
5	15432147	616/582-3906
6	18543489	512/478-0788
7	19783482	703/714-2900
8	19876078	209/686-3953
9	24589689	(012)736-202
10	26422096	4268-54-72
11	26984578	43-57-04
12	27654351	02/215-37-32
13	28710427	(021)570517
14	29834248	(0552)715311
15	31548901	406/422-3413
16	38763919	244-6324
17	39045213	012/302-1021
18	43290587	(02)933-3212
19	43459747	03/734-5111
20	46543295	(03)022-2332
21	46783280	3762855
22	48345514	213445
OBS	NAME	
1		
2	SANTA CLARA VALLEY TECHNOLOGY SPECIALISTS	
3	PRECISION PRODUCTS	
4	UNIVERSITY BIOMEDICAL MATERIALS	
5	GREAT LAKES LABORATORY EQUIPMENT MANUFACTURERS	
6	LONE STAR STATE RESEARCH SUPPLIERS	
7	TWENTY-FIRST CENTURY MATERIALS	
8	SAN JOAQUIN SCIENTIFIC AND INDUSTRIAL SUPPLY, INC.	
9	CENTAR ZA TECHNICKU I NAUCNU RESTAURIRANJE UMJETNINA	
10	SOCIETE DE RECHERCHES POUR DE CHIRURGIE ORTHOPEDIQUE	
11	INSTITUT FUR TEXTIL-FORSCHUNGS	
12	INSTITUT DE RECHERCHE SCIENTIFIQUE MEDICALE	
13	ANTONIE VAN LEEUWENHOEK VERENIGING VOOR MICROBIOLOGIE	
14	BRITISH MEDICAL RESEARCH AND SURGICAL SUPPLY	
15	NATIONAL COUNCIL FOR MATERIALS RESEARCH	
16	INSTITUTO DE BIOLOGIA Y MEDICINA NUCLEAR	
17	LABORATORIO DE PESQUISAS VETERINARIAS DESIDERIO FINAMOR	
18	HASSEI SAIBO GAKKAI	
19	RESEARCH OUTFITTERS	
20	WESTERN TECHNOLOGICAL SUPPLY	
21	NGEE TECHNOLOGICAL INSTITUTE	
22	GULF SCIENTIFIC SUPPLIES	

Output 3.10 Data That is Described by the View Descriptor VLIB.CUSORDR

Data Described by VLIB.CUSORDR		
OBS	STOCKNUM	SHIPTO
1	9870	19876078
2	1279	39045213
3	8934	18543489
4	3478	29834248
5	2567	19783482
6	4789	15432147
7	3478	29834248
8	1279	14324742
9	8934	31548901
10	2567	14898029
11	9870	48345514
12	1279	39045213
13	8934	18543489
14	2567	19783482
15	9870	18543489
16	3478	24589689
17	1279	38763919
18	8934	43459747
19	2567	15432147
20	9870	14324742
21	9870	19876078
22	1279	39045213
23	8934	18543489
24	3478	29834248
25	2567	19783482
26	4789	15432147
27	3478	29834248
28	1279	14324742
29	8934	31548901
30	2567	14898029
31	9870	48345514
32	1279	39045213
33	8934	18543489
34	2567	19783482
35	9870	18543489
36	3478	24589689
37	1279	38763919
38	8934	43459747
39	2567	15432147
40	9870	14324742

Output 3.11 Data in the SAS Data File MYDATA.OUTOFSTK

SAS Data File MYDATA.OUTOFSTK		
OBS	FIBERNAM	FIBERNUM
1	olefin	3478
2	gold	8934
3	dacron	4789

The following SAS code selects and combines data from these three sources to create a PROC SQL view, SQL.BADORDR. The SQL.BADORDR view retrieves customer and product information that the sales department can use to notify customers of unavailable products.

```
proc sql;
create view sql.badordr as
  select cusphon.custnum, cusphon.name,
         cusphon.phone, cusordr.stocknum,
         outofstk.fibernam as product
  from vlib.cusphon, vlib.cusordr,
       mydata.outofstk
  where cusordr.stocknum=outofstk.fibernum
        and cusphon.custnum=cusordr.shipto
  order by cusphon.custnum, product;
title "Data Described by SQL.BADORDR";
select * from sql.badordr;
```

The CREATE VIEW statement incorporates a WHERE clause as part of its SELECT statement. The last SELECT statement retrieves and displays the PROC SQL view, SQL.BADORDR. To select all columns from the view, use an asterisk (*) in place of variable names. The order of the columns displayed matches the order of the columns as specified in the view descriptor SQL.BADORDR. (Note that an ORDER BY clause requires an ADABAS descriptor data field.)

The following output shows the data that is described by the SQL.BADORDR view. Note that the SQL procedure uses the column labels in the output by default.

Output 3.12 Results of Combining Data from Various Sources

Data Described by SQL.BADORDR		
CUSTOMER	NAME	
TELEPHONE	STOCKNUM	PRODUCT

15432147	GREAT LAKES LABORATORY EQUIPMENT MANUFACTURERS	
616/582-3906	4789	dacron
15432147	GREAT LAKES LABORATORY EQUIPMENT MANUFACTURERS	
616/582-3906	4789	dacron
18543489	LONE STAR STATE RESEARCH SUPPLIERS	
512/478-0788	8934	gold
18543489	LONE STAR STATE RESEARCH SUPPLIERS	
512/478-0788	8934	gold
18543489	LONE STAR STATE RESEARCH SUPPLIERS	
512/478-0788	8934	gold
18543489	LONE STAR STATE RESEARCH SUPPLIERS	
512/478-0788	8934	gold
24589689	CENTAR ZA TECHNICKU I NAUCNU RESTAURIRANJE UMJETNINA	
(012)736-202	3478	olefin
24589689	CENTAR ZA TECHNICKU I NAUCNU RESTAURIRANJE UMJETNINA	
(012)736-202	3478	olefin
29834248	BRITISH MEDICAL RESEARCH AND SURGICAL SUPPLY	
(0552)715311	3478	olefin
29834248	BRITISH MEDICAL RESEARCH AND SURGICAL SUPPLY	
(0552)715311	3478	olefin
29834248	BRITISH MEDICAL RESEARCH AND SURGICAL SUPPLY	
(0552)715311	3478	olefin
29834248	BRITISH MEDICAL RESEARCH AND SURGICAL SUPPLY	
(0552)715311	3478	olefin
31548901	NATIONAL COUNCIL FOR MATERIALS RESEARCH	
406/422-3413	8934	gold
31548901	NATIONAL COUNCIL FOR MATERIALS RESEARCH	
406/422-3413	8934	gold
43459747	RESEARCH OUTFITTERS	
03/734-5111	8934	gold
43459747	RESEARCH OUTFITTERS	
03/734-5111	8934	gold

The view SQL.BADORDR lists entries for all customers who have ordered out-of-stock products. However, it contains duplicate rows because some companies have ordered the same product more than once. To make the data more readable for the sales department, you can create a final SAS data file, MYDATA.BADNEWS, using the results of the PROC SQL view as input in the SET statement and the special variable FIRST.PRODUCT. This variable identifies which row is the first in a particular BY group. You only need a customer's name once to notify them that a product is out of stock, regardless of the number of times the customer has placed an order for it.

```

data mydata.badnews;
  set sql.badordr;
  by custnum product;
  if first.product;
run;

proc print;
  title "MYDATA.BADNEWS Data File";
quit;

```

The data file MYDATA.BADNEWS contains an observation for each unique combination of customer and out-of-stock product. The following output displays this data file.

Output 3.13 Results of Grouping Data Using First.variable

MYDATA.BADNEWS Data File			
OBS	CUSTNUM	NAME	
1	15432147	GREAT LAKES LABORATORY EQUIPMENT MANUFACTURERS	
2	18543489	LONE STAR STATE RESEARCH SUPPLIERS	
3	24589689	CENTAR ZA TECHNICKU I NAUCNU RESTAURIRANJE UMJETNINA	
4	29834248	BRITISH MEDICAL RESEARCH AND SURGICAL SUPPLY	
5	31548901	NATIONAL COUNCIL FOR MATERIALS RESEARCH	
6	43459747	RESEARCH OUTFITTERS	
OBS	PHONE	STOCKNUM	PRODUCT
1	616/582-3906	4789	dacron
2	512/478-0788	8934	gold
3	(012)736-202	3478	olefin
4	(0552)715311	3478	olefin
5	406/422-3413	8934	gold
6	03/734-5111	8934	gold

For more information about *FIRST.variable*, see the *SAS Language Reference: Dictionary*.

Creating New Variables with the GROUP BY Clause

It is often useful to create new variables with summarizing or variable functions such as AVG or SUM. Although you cannot use the ACCESS procedure to create new variables, you can easily use the SQL procedure with data that is described by a view descriptor to display output that contains new variables.

This example uses the SQL procedure to retrieve and manipulate data accessed by the view descriptor VLIB.ALLEMP, which accesses data in the NATURAL DDM named EMPLOYEE. When this *query* (as a SELECT statement is often called) is submitted, it calculates and displays the average salary for each department; the AVG function is the SQL procedure's equivalent of the SAS MEAN function.

```

proc sql;
  title "Average Salary Per Department";
  select distinct dept,
    avg(salary) label="Average Salary"
    format=dollar12.2
  from vlib.allemp

```

```

where dept is not missing
group by dept;

```

The order of the variables that are displayed matches the order of the variables as specified in the SELECT list of the query. The following output shows the query's result.

Output 3.14 Results of Creating New Variables With the GROUP BY Clause

Average Salary Per Department	
DEPT	Average Salary
ACC013	\$54,591.33
ACC024	\$55,370.55
ACC043	\$75,000.34
CSR004	\$17,000.00
CSR010	\$44,324.19
CSR011	\$41,966.16
SHP002	\$40,111.31
SHP013	\$41,068.44
SHP024	\$50,000.00

For more information about the SQL procedure, see the SQL section in the *Base SAS Procedures Guide*.

Updating a SAS Data File with ADABAS Data

You can update a SAS data file with ADABAS data that is described by a view descriptor, just as you can update a SAS data file with data from another data file. In this section, the term *transaction data* refers to the new data that is to be added to the original file. You can even do updates when the file to be updated is a Version 6 data file and the transaction data is from a Version 7 and later source.

Suppose you have a Version 6 data file, LIB6.BIRTHDAY, that contains employee ID numbers, last names, and birthdays. You want to update this data file with data that is described by VLIB.EMPS, a view descriptor that is based on the EMPLOYEE DDM. To perform the update, enter the following SAS statements.

```

proc sort data=lib6.birthday;
  by lastname;
run;

proc print data=lib6.birthday;
  title "LIB6.BIRTHDAY Data File";
  format birthdat date7.;
run;

proc print data=vlib.emps;
  title "Data Described by VLIB.EMPS";
run;

data mydata.newbday;
  update lib6.birthday vlib.emps;
  by lastname;
run;

```

```
proc print;
  title 'MYDATA.NEWBDAY Data File';
run;
```

In this example, the new, updated SAS data file, MYDATA.NEWBDAY, is a Version 7 or later data file. It is stored in the Version 7 or later SAS data library associated with the libref MYDATA.

When the UPDATE statement references the view descriptor VLIB.EMPS and uses a BY statement in the DATA step, the BY statement causes a BY clause to be generated for the variable LASTNAME. (Note that a BY statement must reference an ADABAS descriptor data field.) Thus, the BY clause causes the ADABAS data to be presented to SAS in a sorted order for use in updating the MYDATA.NEWBDAY data file. However, the data file LIB6.BIRTHDAY had to be sorted before the update, because the UPDATE statement expects both the original file and the transaction file to be sorted by the BY variable.

The following three outputs show the results of PRINT procedures on the original data file, the transaction data, and the updated data file.

Output 3.15 Data in the Data File to Be Updated, LIB6.BIRTHDAY

LIB6.BIRTHDAY Data File				
OBS	EMPID	BIRTHDAT	LASTNAME	
1	129540	31JUL60	CHOULAI	
2	356134	25OCT60	DUNNETT	
3	127845	25DEC43	MEDER	
4	677890	24APR65	NISHIMATSU-LYNCH	
5	459287	05JAN34	RODRIGUES	
6	346917	15MAR50	SHIEKELESLAN	
7	254896	06APR49	TAYLOR-HUNYADI	

Output 3.16 Data That is Described by the View Descriptor VLIB.EMPS

Data Described by VLIB.EMPS				
OBS	EMPID	JOBCODE	BIRTHDAT	LASTNAME
1	456910	602	24SEP53	ARDIS
2	237642	602	13MAR54	BATTERSBY
3	239185	602	28AUG59	DOS REMEDIOS
4	321783	602	03JUN35	GONZALES
5	120591	602	12FEB46	HAMMERSTEIN
6	135673	602	21MAR61	HEMESLY
7	456921	602	12MAY62	KRAUSE
8	457232	602	15OCT63	LOVELL
9	423286	602	31OCT64	MIFUNE
10	216382	602	24JUL63	PURINTON
11	234967	602	21DEC67	SMITH
12	212916	602	29MAY28	WACHBERGER
13	119012	602	05JAN46	WOLF-PROVENZA

Output 3.17 Results of Updating a Data File with ADABAS Data

MYDATA.NEWBDAY Data File				
OBS	EMPID	BIRTHDAT	LASTNAME	JOBCODE
1	456910	24SEP53	ARDIS	602
2	237642	13MAR54	BATTERSBY	602
3	129540	31JUL60	CHOULAI	.
4	239185	28AUG59	DOS REMEDIOS	602
5	356134	25OCT60	DUNNETT	.
6	321783	03JUN35	GONZALES	602
7	120591	12FEB46	HAMMERSTEIN	602
8	135673	21MAR61	HEMESLY	602
9	456921	12MAY62	KRAUSE	602
10	457232	15OCT63	LOVELL	602
11	127845	25DEC43	MEDER	.
12	423286	31OCT64	MIFUNE	602
13	677890	24APR65	NISHIMATSU-LYNCH	.
14	216382	24JUL63	PURINTON	602
15	459287	05JAN34	RODRIGUES	.
16	346917	15MAR50	SHIEKELESLAN	.
17	234967	21DEC67	SMITH	602
18	254896	06APR49	TAYLOR-HUNYADI	.
19	212916	29MAY28	WACHBERGER	602
20	119012	05JAN46	WOLF-PROVENZA	602

For more information about the UPDATE statement, see *SAS Language Reference: Dictionary*.

Note: You cannot update ADABAS data directly using the DATA step, but you can update ADABAS data using the following procedures: APPEND, FSEDIT, FSVIEW, and SQL. For more information about updating ADABAS data, see Chapter 4, “Browsing and Updating ADABAS Data,” on page 37. \triangle

Performance Considerations

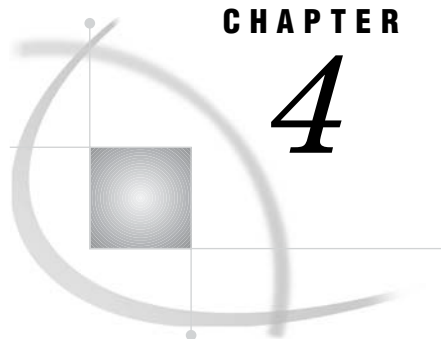
While you can generally treat view descriptors like other SAS data sets in SAS programs, here are a few things you should keep in mind:

- It is sometimes better to extract ADABAS data and place it in a SAS data file rather than to read it directly. Here are some circumstances when you should probably extract:
 - If you plan to use the same ADABAS data in several procedures during the same SAS session, you might improve performance by extracting the ADABAS data. Placing this data in a SAS data file requires a certain amount of disk space to store the data and I/O to write the data. However, SAS data files are organized to provide optimal performance with PROC and DATA steps. Programs using SAS data files often use less CPU time than programs that directly read ADABAS data.
 - If you plan to read large amounts of ADABAS data and the data is being shared by several users, your direct reading of the data could adversely affect all users' response time.
 - If you are the creator of an ADABAS file and think that directly reading this data would present a security risk, you might want to extract the data and not distribute information about either the access descriptor or view descriptor.

- If you intend to use the data in a particular sorted order several times, it is usually best to run the SORT procedure on the view descriptor, using the OUT= option. This is more efficient than requesting the same sort repeatedly (with a BY clause) on the ADABAS data. Note that you cannot run the SORT procedure on a view descriptor unless you use the SORT procedure's OUT= option.
- Sorting data can be resource-intensive, whether it is done with the SORT procedure, with a BY statement (which generates a BY clause), or with a SORT clause stored in the view descriptor. You should sort data only when it is needed for your program.
- If you reference a view descriptor in SAS code and the code includes a BY statement for a variable or variables (up to three) that corresponds to a descriptor data field in the ADABAS file, the interface view engine is called, and it will support the BY clause if possible. Thus, the BY clause sorts the ADABAS data before it uses the data in your SAS program. If the ADABAS file is very large, this sorting can affect performance.

If the view descriptor already has a SORT clause and you specify a BY statement in your SAS code, the BY statement overrides the view descriptor's SORT clause.

- When writing a SAS program and referencing a view descriptor, it is more efficient to use a SAS WHERE statement in the program than it is to use a subsetting IF statement. The SAS program passes the WHERE statement as a WHERE clause to the interface view engine, which adds it (using the Boolean operator AND) to any WHERE clause stored in the view descriptor. The view descriptor is then passed to ADABAS for processing. Applying a WHERE clause to the ADABAS data might reduce the number of logical records read; therefore, it often improves performance.
- Refer to "Creating and Using ADABAS View Descriptors Efficiently" on page 94 for more details about creating efficient view descriptors.



CHAPTER

4

Browsing and Updating ADABAS Data

<i>Introduction to Browsing and Updating ADABAS Data</i>	37
<i>Browsing and Updating ADABAS Data with the SAS/FSP Procedures</i>	38
<i>Browsing Data Using the FSBROWSE Procedure</i>	38
<i>Updating Data Using the FSEDIT Procedure</i>	38
<i>Browsing and Updating Data Using the FSVIEW Procedure</i>	38
<i>Browsing Data Using the FSVIEW Procedure</i>	38
<i>Updating Data Using the FSVIEW Procedure</i>	39
<i>Specifying a SAS WHERE Expression While Browsing or Updating Data</i>	39
<i>Adding and Deleting Data with the SAS/FSP Procedures</i>	41
<i>Adding Data</i>	41
<i>Deleting Data</i>	41
<i>Browsing and Updating ADABAS Data with the SQL Procedure</i>	43
<i>Browsing Data with the SELECT Statement</i>	43
<i>Updating Data with the UPDATE Statement</i>	45
<i>Inserting and Deleting Data with the INSERT and DELETE Statements</i>	47
<i>Appending ADABAS Data with the APPEND Procedure</i>	49

Introduction to Browsing and Updating ADABAS Data

The SAS/ACCESS interface to ADABAS enables you to browse and update ADABAS data directly from a SAS session or program. This section shows you how to use SAS procedures to browse and update ADABAS data that is described by SAS/ACCESS view descriptors.

For definitions of the view descriptors used in this section as well as their associated access descriptors, and the ADABAS files, NATURAL DDMs, and SAS data files used throughout the document, see Appendix 3, “Example Data,” on page 131.

Before you can browse or update ADABAS data, you must have access to the data through appropriate security options. ADABAS and NATURAL have several levels of security options, and you might be allowed to display or browse data but not update values. Check with your Database Administrator (DBA) or the ADABAS file’s or NATURAL DDM’s creator to see what security options you have. If you have been granted the appropriate ADABAS security options, you can use the SAS procedures described in this section to update ADABAS data with a SAS/ACCESS view descriptor. For more information about ADABAS and NATURAL security, see Chapter 2, “ADABAS Essentials,” on page 7, and Appendix 1, “Information for the Database Administrator,” on page 101.

Browsing and Updating ADABAS Data with the SAS/FSP Procedures

If your site has SAS/FSP software as well as SAS/ACCESS software, you can browse and update ADABAS data from within a SAS program.

You can use three SAS/FSP procedures: FSBROWSE, FSEDIT, and FSVIEW. The FSBROWSE and FSEDIT procedures show you one ADABAS logical record at a time, whereas the FSVIEW procedure displays multiple logical records in a tabular format similar to the PRINT procedure. PROC FSVIEW enables you to both browse and update ADABAS data, depending on which option you choose.

Browsing Data Using the FSBROWSE Procedure

The FSBROWSE procedure enables you to look at ADABAS data that is described by a view descriptor but does not enable you to change it. For example, the following SAS statements enable you to view one record at a time of the view descriptor VLIB.USACUST:

```
proc fsbrowse data=vlib.usacust;
run;
```

The FSBROWSE procedure retrieves one logical record of ADABAS data at a time. To browse each logical record, issue the FORWARD and BACKWARD commands.

Updating Data Using the FSEDIT Procedure

The FSEDIT procedure enables you to update ADABAS data that is described by a view descriptor if you have access to the data through the appropriate ADABAS and NATURAL security options. For example, the following SAS statements enable you to browse one record of VLIB.USACUST at a time:

```
proc fsedit data=vlib.usacust;
run;
```

A window similar to the FSBROWSE window opens to enable you to edit the ADABAS data one observation at a time.

Note: When using PROC FSEDIT, you can cancel an edit only before you scroll. The CANCEL command redisplay the observation as it was before you began to edit it and cancels your editing changes. After you scroll, the changes are saved. Δ

Browsing and Updating Data Using the FSVIEW Procedure

The FSVIEW procedure enables you to browse or update ADABAS data using a view descriptor, depending on how you submit the procedure.

Browsing Data Using the FSVIEW Procedure

Browse mode is the default for the FSVIEW procedure. For example, to browse ADABAS data, submit the PROC FSVIEW statement as follows:

```
proc fsview data=vlib.usacust;
run;
```

The statements display the data as shown in the following output.

Output 4.1 Results of Browsing Data Using the FSVIEW Procedure

```

FSVIEW: VLIB.USACUST (B)
Command ==>
  ROW      CUSTNUM    STATE  ZIPCODE  COUNTRY
    1      12345678   NC      27702   USA
    2      14324742   CA      95123   USA
    3      14324742   CA      95123   USA
    4      14569877   NC      27514   USA
    5      14569877   NC      27514   USA
    6      14898029   MD      20850   USA
    7      14898029   MD      20850   USA
    8      14898029   MD      20850   USA
    9      15432147   MI      49001   USA
   10      18543489   TX      78701   USA
   11      18543489   TX      78701   USA
   12      18543489   TX      78701   USA
   13      19783482   VA      22090   USA
   14      19783482   VA      22090   USA
   15      19876078   CA      93274   USA
   16      19876078   CA      93274   USA

```

To see the rest of the accessed ADABAS data, you must scroll the window to the right multiple times. You can do this by entering the RIGHT command on the command line or by pressing the function key assigned to this command.

Updating Data Using the FSVIEW Procedure

You can use the FSVIEW procedure to update ADABAS data. To edit the ADABAS data in a listing format, you have to add EDIT or MODIFY to the PROC FSVIEW statement, as shown in the following statement:

```

proc fsview data=vlib.usacust edit;
run;

```

The same window as shown in Output 4.1 appears, except the window title contains an (E) for edit, not a (B). *SAS/FSP Software: Usage and Reference* discusses in detail how to edit data using the FSVIEW procedure.

Note: The CANCEL command in the FSVIEW window does *not* cancel your changes, whether you have scrolled or not. Δ

Specifying a SAS WHERE Expression While Browsing or Updating Data

You can specify a SAS WHERE statement or a SAS WHERE command to retrieve a subset of ADABAS data while you are using the FSP procedures. The WHERE statement is submitted when the FSP procedure is invoked and retrieves only the observations that meet the conditions of the WHERE statement. The other observations are not available until you exit the procedure. This is called a *permanent* WHERE clause. A SAS WHERE command is a WHERE expression that is invoked from the command line within a FSP procedure. You can clear the command to make all the observations available so it is known as a *temporary* WHERE clause.

The following example of a WHERE statement retrieves the customers from California. These customers are a subset of the customers for the CUSTOMERS DDM.

```
proc fsview data=vlib.usacust edit;
  where state='CA';
run;
```

The following output shows the FSVIEW window after the statements have been submitted.

Output 4.2 Results of Specifying a WHERE Statement While Updating Data

FSVIEW: VLIB.USACUST (Subset)				
Command ==>				
ROW	CUSTNUM	STATE	ZIPCODE	COUNTRY
2	14324742	CA	95123	USA
3	14324742	CA	95123	USA
15	19876078	CA	93274	USA
16	19876078	CA	93274	USA

Only the logical records with a STATE value of **CA** are retrieved for editing. Note that (Subset) appears after VLIB.USACUST in the window title to remind you that the data retrieved is a subset of the data that is described by the view descriptor. You can then edit each observation by typing over the information you want to modify. Issue the END command to end your editing session.

The following output shows the FSVIEW window when the subset of data is generated by the WHERE command:

```
where state='CA'
```

Output 4.3 Results of Specifying a WHERE Command While Updating Data

FSVIEW VLIB.USACUST				WHERE ...
Command ==>				
ROW	CUSTNUM	STATE	ZIPCODE	COUNTRY
2	14324742	CA	95123	USA
3	14324742	CA	95123	USA
15	19876078	CA	93274	USA
16	19876078	CA	93274	USA

Output 4.2 and Output 4.3 are identical, except (Subset) after the title is replaced with **WHERE** in the upper-right corner. You can then update each observation, as described earlier.

Although these examples have shown a SAS WHERE statement and a SAS WHERE command with the FSVIEW procedure, you can also retrieve a subset of data using the FSBROWSE and FSEDIT procedures. For more information about the SAS WHERE statement, refer to *SAS Language Reference: Dictionary*. For more information about using the SAS WHERE command within the SAS/FSP procedures, refer to *SAS/FSP Software: Usage and Reference*.

Adding and Deleting Data with the SAS/FSP Procedures

Adding and deleting ADABAS data with the SAS/FSP procedures is different for view descriptors than for SAS data files.

Adding Data

Adding ADABAS data as a result of any SAS update operation can cause the interface view engine to decide whether to add a new ADABAS logical record or to modify an existing one, for example, to add an occurrence to a periodic group.

If there are no periodic group fields accessed by the view descriptor or within the ADABAS file, doing an insert is straightforward. However, if a periodic group does exist, then doing an insert is more complicated, because the interface view engine generates multiple SAS observations from a single ADABAS record that contains a periodic group.

Values in the observation to be added are compared to values in the previous observation. If the contents of the previous observation do not help determine whether to add or modify, a new logical record is added. However, it is possible that some of the new values might already reside in the ADABAS file, which would mean that a new logical record is not necessary. This occurs if a periodic group is selected by the view descriptor, and the new data occurs only in variables corresponding to data fields that are part of that periodic group.

You can help the interface view engine resolve whether to add a new logical record or modify an existing one by specifying BY keys. For information about and examples of using BY keys, see “Using a BY Key To Resolve Ambiguous Inserts” on page 119.

Deleting Data

When you delete a logical record, the results depend on whether the observation is part of a periodic group. If the logical record is not part of a periodic group, deleting an observation causes a logical record to be deleted from the ADABAS file. However, if the logical record is part of a periodic group, the results of deleting an observation depend on the status of the ADBDEL systems option for the interface view engine, which is set in the ADBEUSE CSECT. For more information, see “System Options for PROC ACCESS and the Interface View Engine” on page 112.

- If ADBDEL=N (which is the default setting), the selected values for that occurrence in the periodic group are set to null (missing), but the logical record is not deleted.
- If ADBDEL=P, the entire logical record is deleted.

The following example illustrates using the DELETE command in the FSEDIT procedure. (Note that the ADBDEL systems option is set to N.)

Suppose you want to edit the ADABAS data described by VLIB.USACUST. You can use the FSEDIT procedure with a PROC FSEDIT statement. Scroll forward to the observation to be deleted. In this example, there are three occurrences for the periodic group SIGNATURE-LIST. The following output shows the third occurrence, which you want to delete. (Notice that the variable SL_OCCUR displays the value 3, which tells you that this is the observation for the third occurrence.) Enter the DELETE command on the command line, as shown in the following output, and press ENTER.

Output 4.4 Results of Deleting an ADABAS Logical Record

```

FSEDIT VLIB.USACUST
Command ==> delete
  CUSTNUM: 18543489

  STATE: TX

  ZIPCODE: 78701

  COUNTRY: USA

  NAME: LONE STAR STATE RESEARCH SUPPLIERS

  FIRSTORD: 10SEP79

  SL_OCCUR: 3

  LIMIT: 100000.00

  SIGNATUR: EVAN MASSEY

  BRANCH_2: DALLAS

```

The DELETE command processes the deletion and displays a message to that effect, as shown in the following output. There is no indication of what actions the interface view engine actually took.

Output 4.5 Deletion Message Displayed

```

FSEDIT VLIB.USACUST                                DELETED
Command ==>
NOTE: Observation has been deleted.
  CUSTNUM: _____
  STATE:  __

  ZIPCODE: _____

  COUNTRY: _____

  NAME:  _____

  FIRSTORD: _____

  SL_OCCUR: _____

  LIMIT:  _____

  SIGNATUR: _____

  BRANCH_2: _____

```

The entire observation seems to have been removed from the ADABAS file, but this is not the case. For the third occurrence, the interface view engine sets the values for data fields LIMIT and SIGNATUR to missing; the other data remains the same. Regardless of the actions though, the observation you deleted is no longer available for processing. For more information about using the SAS/FSP procedures, see *SAS/FSP Software: Usage and Reference*.

Browsing and Updating ADABAS Data with the SQL Procedure

The SAS SQL procedure enables you to retrieve and update ADABAS data. You can retrieve and browse ADABAS data by specifying a view descriptor in a PROC SQL SELECT statement.

To update the data, you can specify view descriptors in the PROC SQL DELETE, INSERT, and UPDATE statements. You must have access to the data through appropriate ADABAS and NATURAL security options before you can edit ADABAS data. Here is a summary of the pertinent PROC SQL statements:

DELETE	deletes logical records from an ADABAS file.
INSERT	inserts logical records in an ADABAS file.
SELECT	retrieves and displays data from an ADABAS file. A SELECT statement is usually referred to as a query because it queries the ADABAS file for information.
UPDATE	updates values in an ADABAS file.

When using the SQL procedure, note that the data is displayed in the SAS OUTPUT window. The procedure displays output data automatically without using the PRINT procedure and executes without using the RUN statement when an SQL procedure statement is executed.

Browsing Data with the SELECT Statement

You can use the SELECT statement to browse ADABAS data that is described by a view descriptor. The query in the following example retrieves and displays specified data fields and logical records in the CUSTOMERS DDM that are described by the VLIB.USACUST view descriptor. The LINESIZE= system option is used to reset the default output width to 120 columns.

Note: The following SQL procedure examples assume that the CUSTOMERS DDM has not been updated by the earlier SAS/FSP examples. △

```
options linesize=120;

proc sql;
  title 'ADABAS Data Output by a
        SELECT Statement';
  select custnum, state, name, limit,signatur
  from vlib.usacust;
```

The following output displays the query's results. Notice in the output that the SQL procedure displays the ADABAS data field names, not the corresponding SAS variable names.

Output 4.6 Results of Browsing Data with a PROC SQL Query

ADABAS Data Output by a SELECT Statement				
CUSTOMER	STATE	NAME		LIMIT
SIGNATURE				

12345678	NC			0.00
14324742	CA	SANTA CLARA VALLEY TECHNOLOGY SPECIALISTS		5000.00
BOB HENSON				
14324742	CA	SANTA CLARA VALLEY TECHNOLOGY SPECIALISTS		25000.00
KAREN DRESSER				
14569877	NC	PRECISION PRODUCTS		5000.00
JEAN CRANDALL				
14569877	NC	PRECISION PRODUCTS		100000.00
STEVE BLUNTSEN				
14898029	MD	UNIVERSITY BIOMEDICAL MATERIALS		10000.00
MASON FOXWORTH				
14898029	MD	UNIVERSITY BIOMEDICAL MATERIALS		50000.00
DANIEL STEVENS				
14898029	MD	UNIVERSITY BIOMEDICAL MATERIALS		100000.00
ELIZABETH PATTON				
15432147	MI	GREAT LAKES LABORATORY EQUIPMENT MANUFACTURERS		10000.00
JACK TREVANE				
18543489	TX	LONE STAR STATE RESEARCH SUPPLIERS		10000.00
NANCY WALSH				
18543489	TX	LONE STAR STATE RESEARCH SUPPLIERS		50000.00
TED WHISTLER				
18543489	TX	LONE STAR STATE RESEARCH SUPPLIERS		100000.00
EVAN MASSEY				
19783482	VA	TWENTY-FIRST CENTURY MATERIALS		5000.00
PETER THOMAS				
19783482	VA	TWENTY-FIRST CENTURY MATERIALS		10000.00
LOUIS PICKERING				
19876078	CA	SAN JOAQUIN SCIENTIFIC AND INDUSTRIAL SUPPLY, INC.		7500.00
EDWARD LOWE				
19876078	CA	SAN JOAQUIN SCIENTIFIC AND INDUSTRIAL SUPPLY, INC.		25000.00
E.F. JENSEN				

You can specify a WHERE clause as part of the SELECT statement to retrieve a subset of the logical records for display. The following example displays the companies that are located in North Carolina:

```
title 'ADABAS Data Output by a WHERE Clause';
select custnum, state, name, limit, signatur
  from vlib.usacust
  where state='NC';
```

Notice that the PROC SQL statement is not repeated in this query. With the SQL procedure, you do not need to repeat the PROC SQL statement unless you use another

SAS procedure, a DATA step, or a QUIT statement between PROC SQL statements. The following output displays the companies from North Carolina described by VLIB.USACUST.

Output 4.7 Results of Browsing Data Subset by a WHERE Clause

ADABAS Data Output by a WHERE Clause			
CUSTOMER	STATE	NAME	LIMIT
SIGNATURE			

12345678	NC		0.00
14569877	NC	PRECISION PRODUCTS	5000.00
JEAN CRANDALL			
14569877	NC	PRECISION PRODUCTS	100000.00
STEVE BLUNTSSEN			

Updating Data with the UPDATE Statement

You can use the UPDATE statement to update ADABAS data. Remember that when you reference a view descriptor in a PROC SQL statement, you are not updating the view descriptor, but rather the ADABAS data described by the view descriptor.

The following UPDATE statements update the values described by the logical record that meets the WHERE clause criteria. The SELECT statement then displays the view's output as shown in Output 4.8 . The ORDER BY clause in the SELECT statement causes the data to be presented in ascending order by the CUSTOMER data field. (Because you are referencing a view descriptor, you use the SAS variable names for data fields in the UPDATE statement; however, the SQL procedure displays the ADABAS data field names.)

```
update vlib.usacust
  set zipcode=27702
  where custnum='12345678';
update vlib.usacust
  set name='DURHAM SCIENTIFIC SUPPLY COMPANY'
  where custnum='12345678';
update vlib.usacust
  set firstord='02JAN88'd
  where custnum='12345678';
update vlib.usacust
  set limit=5000.00
  where custnum='12345678';
update vlib.usacust
  set signatur='MARC PLOUGHMAN'
  where custnum='12345678';
update vlib.usacust
  set branch_2='DURHAM'
  where custnum='12345678';
title 'Updated ADABAS Data in CUSTOMERS';
```

```
select custnum, state, name, limit, signatur
from vlib.usacust;
```

Output 4.8 Results of Updating Data with the UPDATE Statement

Updated ADABAS Data in CUSTOMERS			
CUSTOMER SIGNATURE	STATE	NAME	LIMIT
12345678 MARC PLOUGHMAN	NC	DURHAM SCIENTIFIC SUPPLY COMPANY	5000.00
14324742 BOB HENSON	CA	SANTA CLARA VALLEY TECHNOLOGY SPECIALISTS	5000.00
14324742 KAREN DRESSER	CA	SANTA CLARA VALLEY TECHNOLOGY SPECIALISTS	25000.00
14569877 JEAN CRANDALL	0 NC	PRECISION PRODUCTS	5000.00
14569877 STEVE BLUNTSEN	NC	PRECISION PRODUCTS	100000.00
14898029 MASON FOXWORTH	MD	UNIVERSITY BIOMEDICAL MATERIALS	10000.00
14898029 DANIEL STEVENS	MD	UNIVERSITY BIOMEDICAL MATERIALS	50000.00
14898029 ELIZABETH PATTON	MD	UNIVERSITY BIOMEDICAL MATERIALS	100000.00
15432147 JACK TREVANE	MI	GREAT LAKES LABORATORY EQUIPMENT MANUFACTURERS	10000.00
18543489 NANCY WALSH	TX	LONE STAR STATE RESEARCH SUPPLIERS	10000.00
18543489 TED WHISTLER	TX	LONE STAR STATE RESEARCH SUPPLIERS	50000.00
18543489 EVAN MASSEY	TX	LONE STAR STATE RESEARCH SUPPLIERS	100000.00
19783482 PETER THOMAS	VA	TWENTY-FIRST CENTURY MATERIALS	5000.00
19783482 LOUIS PICKERING	VA	TWENTY-FIRST CENTURY MATERIALS	10000.00
19876078 EDWARD LOWE	CA	SAN JOAQUIN SCIENTIFIC AND INDUSTRIAL SUPPLY, INC.	7500.00
19876078 E.F. JENSEN	CA	SAN JOAQUIN SCIENTIFIC AND INDUSTRIAL SUPPLY, INC.	25000.00

Inserting and Deleting Data with the INSERT and DELETE Statements

You can use the INSERT statement to add logical records to an ADABAS file or the DELETE statement to remove logical records. In the following example, the logical record containing the CUSTOMER value 15432147 is deleted by using the CUSTOMERS DDM. The SELECT statement then displays the VLIB.USACUST data in the following output, ordering them again by the CUSTOMER data field.

```
delete from vlib.usacust
  where custnum='15432147';
  title 'Logical Record Deleted from
    CUSTOMERS';
select custnum, state, name, limit, signatur
  from vlib.usacust;
```

Output 4.9 Results of Deleting Data with the DELETE Statement

Updated ADABAS Data in CUSTOMERS			
CUSTOMER SIGNATURE	STATE	NAME	LIMIT
12345678 MARC PLOUGHMAN	NC	DURHAM SCIENTIFIC SUPPLY COMPANY	5000.00
14324742 BOB HENSON	CA	SANTA CLARA VALLEY TECHNOLOGY SPECIALISTS	5000.00
14324742 KAREN DRESSER	CA	SANTA CLARA VALLEY TECHNOLOGY SPECIALISTS	25000.00
14569877 JEAN CRANDALL	NC	PRECISION PRODUCTS	5000.00
14569877 STEVE BLUNTSEN	NC	PRECISION PRODUCTS	100000.00
14898029 MASON FOXWORTH	MD	UNIVERSITY BIOMEDICAL MATERIALS	10000.00
14898029 DANIEL STEVENS	MD	UNIVERSITY BIOMEDICAL MATERIALS	50000.00
14898029 ELIZABETH PATTON	MD	UNIVERSITY BIOMEDICAL MATERIALS	100000.00
18543489 NANCY WALSH	TX	LONE STAR STATE RESEARCH SUPPLIERS	10000.00
18543489 TED WHISTLER	TX	LONE STAR STATE RESEARCH SUPPLIERS	50000.00
18543489 EVAN MASSEY	TX	LONE STAR STATE RESEARCH SUPPLIERS	100000.00
19783482 PETER THOMAS	VA	TWENTY-FIRST CENTURY MATERIALS	5000.00
19783482 LOUIS PICKERING	VA	TWENTY-FIRST CENTURY MATERIALS	10000.00
19876078 EDWARD LOWE	CA	SAN JOAQUIN SCIENTIFIC AND INDUSTRIAL SUPPLY, INC.	7500.00
19876078 E.F. JENSEN	CA	SAN JOAQUIN SCIENTIFIC AND INDUSTRIAL SUPPLY, INC.	25000.00

CAUTION:

Always use the WHERE clause in a DELETE statement. If you omit the WHERE clause from a DELETE statement, you delete *all* the data in the ADABAS file that is accessed by the view descriptor. Δ

For more information about SAS SQL procedure, see the SQL section in the *Base SAS Procedures Guide*.

Appending ADABAS Data with the APPEND Procedure

In earlier releases of SAS, the APPEND procedure operated only on SAS data files. You can append data that is described by SAS/ACCESS view descriptors and PROC SQL views to SAS data files and vice versa. You can also append data that is described by view descriptors to each other.

In the following example, two personnel managers have kept separate employee records. One manager has kept records in a NATURAL DDM named EMPLOYEE, which is described by the view descriptor VLIB.ADAEMPS. The other manager has kept records in a SAS data file that is named MYDATA.SASEMPS. Due to a corporate reorganization, the two sources of data must be combined so that all employee data is stored in the EMPLOYEE DDM. The APPEND procedure can do this.

The data that is described by the view descriptor VLIB.ADAEMPS and the data that is in the SAS data file MYDATA.SASEMPS is printed with the following statements and displayed in the following two outputs:

```
options linesize=80;

proc print data=vlib.adaemps;
  title 'Data Described by VLIB.ADAEMPS';
run;

proc print data=mydata.sasemps;
  format birthdat date7.;
  title 'Data in MYDATA.SASEMPS Data File';
run;
```

Output 4.10 Data That Is Described by VLIB.ADAEMPS

Data Described by VLIB.ADAEMPS					
OBS	EMPID	BIRTHDAT	LASTNAME	FIRSTNAM	MIDDLENA
1	119012	05JAN46	WOLF-PROVENZA	G.	ANDREA
2	120591	12FEB46	HAMMERSTEIN	S.	RACHAEL
3	123456	.	VARGAS	PAUL	JESUS
4	127845	25DEC43	MEDER	VLADIMIR	JORAN
5	129540	31JUL60	CHOLAI	CLARA	JANE
6	135673	21MAR61	HEMESLY	STEPHANIE	J.
7	212916	29MAY28	WACHBERGER	MARIE-LOUISE	TERESA
8	216382	24JUL63	PURINTON	PRUDENCE	VALENTINE
9	234967	21DEC67	SMITH	GILBERT	IRVINE
10	237642	13MAR54	BATTERSBY	R.	STEPHEN
11	239185	28AUG59	DOS REMEDIOS	LEONARD	WESLEY
12	254896	06APR49	TAYLOR-HUNYADI	ITO	MISHIMA
13	321783	03JUN35	GONZALES	GUILLELMO	RICARDO
14	328140	02JUN51	MEDINA-SIDONIA	MARGARET	ROSE
15	346917	15MAR50	SHIEKELESLAM	SHALA	Y.
16	356134	25OCT60	DUNNETT	CHRISTINE	MARIE
17	423286	31OCT64	MIFUNE	YUKIO	TOSHIRO
18	456910	24SEP53	ARDIS	RICHARD	BINGHAM
19	456921	12MAY62	KRAUSE	KARL-HEINZ	G.
20	457232	15OCT63	LOVELL	WILLIAM	SINCLAIR
21	459287	05JAN34	RODRIGUES	JUAN	M.
22	677890	24APR65	NISHIMATSU-LYNCH	CAROL	ANNE

Output 4.11 Data in MYDATA.SASEMPS

Data in MYDATA.SASEMPS Data File					
OBS	EMPID	BIRTHDAT	LASTNAME	FIRSTNAM	MIDDLENA
1	245962	30AUG64	BEDORTHA	KATHY	MARTHA
2	765432	01MAR59	POWELL	FRANK	X.
3	219223	13JUN47	HANSINGER	BENJAMIN	HAROLD
4	326745	21FEB52	RAWN	BEATRICE	MAY

The following statements use the APPEND procedure to combine the data from these two sources:

```
proc append base=vlib.adaemps
  data=mydata.sasemps;
run;

proc print data=vlib.adaemps;
  title 'Appended Data';
run;
```

The following output displays the appended data that is described by the view descriptor VLIB.ADAEMPS. Notice that the data in MYDATA.SASEMPS follows the data that is described by VLIB.ADAEMPS.

Output 4.12 Results of Appending Data with the APPEND Procedure

Appended Data					
OBS	EMPID	BIRTHDAT	LASTNAME	FIRSTNAM	MIDDLENA
1	119012	05JAN46	WOLF-PROVENZA	G.	ANDREA
2	120591	12FEB46	HAMMERSTEIN	S.	RACHAEL
3	123456	.	VARGAS	PAUL	JESUS
4	127845	25DEC43	MEDER	VLADIMIR	JORAN
5	129540	31JUL60	CHOU LAI	CLARA	JANE
6	135673	21MAR61	HEMESLY	STEPHANIE	J.
7	212916	29MAY28	WACHBERGER	MARIE-LOUISE	TERESA
8	216382	24JUL63	PURINTON	PRUDENCE	VALENTINE
9	234967	21DEC67	SMITH	GILBERT	IRVINE
10	237642	13MAR54	BATTERSBY	R.	STEPHEN
11	239185	28AUG59	DOS REMEDIOS	LEONARD	WESLEY
12	254896	06APR49	TAYLOR-HUNYADI	ITO	MISHIMA
13	321783	03JUN35	GONZALES	GUILLERMO	RICARDO
14	328140	02JUN51	MEDINA-SIDONIA	MARGARET	ROSE
15	346917	15MAR50	SHIEKELESLAM	SHALA	Y.
16	356134	25OCT60	DUNNETT	CHRISTINE	MARIE
17	423286	31OCT64	MIFUNE	YUKIO	TOSHIRO
18	456910	24SEP53	ARDIS	RICHARD	BINGHAM
19	456921	12MAY62	KRAUSE	KARL-HEINZ	G.
20	457232	15OCT63	LOVELL	WILLIAM	SINCLAIR
21	459287	05JAN34	RODRIGUES	JUAN	M.
22	677890	24APR65	NISHIMATSU-LYNCH	CAROL	ANNE
23	245962	30AUG64	BEDORTHA	KATHY	MARTHA
24	765432	01MAR59	POWELL	FRANK	X.
25	219223	13JUN47	HANSINGER	BENJAMIN	HAROLD
26	326745	21FEB52	RAWN	BEATRICE	MAY

The APPEND procedure also accepts a WHERE= data set option or a SAS WHERE statement to retrieve a subset of the data. In the following example, a subset of the observations from MYDATA.SASEMPS is added to VLIB.ADAEMPS.

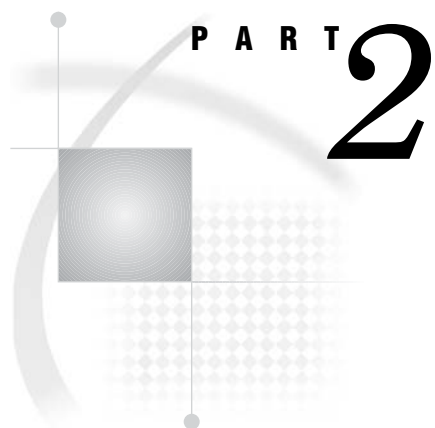
```
proc append base=vlib.adaemps
  data=mydata.sasemps
    (where=(birthdat>='01JAN60'd));
run;
proc print data=vlib.adaemps;
  title 'Appended Data with a WHERE= Data Set
  Option';
run;
```

The results are displayed in the following output.

Output 4.13 Results of Appending Data with a WHERE= Data Set Option

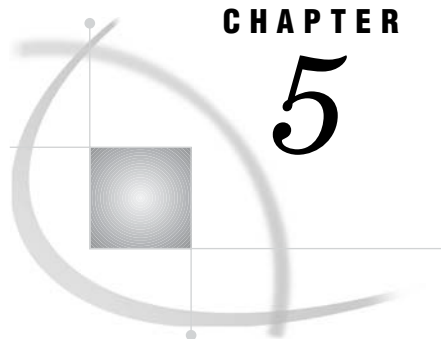
Appended Data with a WHERE= Data Set Option					
OBS	EMPID	BIRTHDAT	LASTNAME	FIRSTNAM	MIDDLENA
1	119012	05JAN46	WOLF-PROVENZA	G.	ANDREA
2	120591	12FEB46	HAMMERSTEIN	S.	RACHAEL
3	123456	.	VARGAS	PAUL	JESUS
4	127845	25DEC43	MEDER	VLADIMIR	JORAN
5	129540	31JUL60	CHOULAI	CLARA	JANE
6	135673	21MAR61	HEMESLY	STEPHANIE	J.
7	212916	29MAY28	WACHBERGER	MARIE-LOUISE	TERESA
8	216382	24JUL63	PURINTON	PRUDENCE	VALENTINE
9	234967	21DEC67	SMITH	GILBERT	IRVINE
10	237642	13MAR54	BATTERSBY	R.	STEPHEN
11	239185	28AUG59	DOS REMEDIOS	LEONARD	WESLEY
12	254896	06APR49	TAYLOR-HUNYADI	ITO	MISHIMA
13	321783	03JUN35	GONZALES	GUILLERMO	RICARDO
14	328140	02JUN51	MEDINA-SIDONIA	MARGARET	ROSE
15	346917	15MAR50	SHIEKELESLAM	SHALA	Y.
16	356134	25OCT60	DUNNETT	CHRISTINE	MARIE
17	423286	31OCT64	MIFUNE	YUKIO	TOSHIRO
18	456910	24SEP53	ARDIS	RICHARD	BINGHAM
19	456921	12MAY62	KRAUSE	KARL-HEINZ	G.
20	457232	15OCT63	LOVELL	WILLIAM	SINCLAIR
21	459287	05JAN34	RODRIGUES	JUAN	M.
22	677890	24APR65	NISHIMATSU-LYNCH	CAROL	ANNE
23	245962	30AUG64	BEDORTHA	KATHY	MARTHA

For more information about the APPEND procedure, see the *Base SAS Procedures Guide*.



SAS/ACCESS Interface to ADABAS: Reference

Chapter 5 **ACCESS Procedure Reference** 55



CHAPTER

5

ACCESS Procedure Reference

<i>Introduction to ACCESS Procedure Reference</i>	55
<i>Case Sensitivity in the ACCESS Procedure</i>	56
<i>ACCESS Procedure Syntax for ADABAS</i>	56
<i>Description</i>	57
<i>PROC ACCESS Statement Options</i>	58
<i>SAS Passwords for SAS/ACCESS Descriptors</i>	58
<i>Assigning Passwords</i>	59
<i>Assigning Passwords with the DATASETS Procedure</i>	59
<i>Invoking the ACCESS Procedure</i>	60
<i>ACCESS PROCEDURE Statements for ADABAS</i>	61
<i>WHERE Clause in an ADABAS View Descriptor</i>	88
<i>View WHERE Clause Syntax</i>	88
<i>View WHERE Clause Examples</i>	89
<i>Specifying Conditions with the SPANS Operator</i>	90
<i>Specifying Expressions</i>	90
<i>Specifying Values in Character Fields</i>	90
<i>Specifying Numeric Format Values</i>	90
<i>Specifying Dates</i>	91
<i>Specifying Values in Superdescriptor Fields</i>	91
<i>Specifying Values in Subdescriptor Fields</i>	92
<i>Specifying Values in Multiple-Value Fields</i>	93
<i>Specifying Values in Periodic Group Fields</i>	93
<i>SORT Clause in a View Descriptor</i>	93
<i>View SORT Clause Syntax</i>	93
<i>SORT Clause Examples</i>	94
<i>Creating and Using ADABAS View Descriptors Efficiently</i>	94
<i>ACCESS Procedure Formats and Informats for ADABAS</i>	95
<i>Effects of the SAS/ACCESS Interface on ADABAS Data</i>	97

Introduction to ACCESS Procedure Reference

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

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

- “Creating and Using ADABAS View Descriptors Efficiently” on page 94 presents efficiency considerations for using the SAS/ACCESS interface to ADABAS.

- “ACCESS Procedure Formats and Informats for ADABAS” on page 95 summarizes how the SAS/ACCESS interface converts each type of ADABAS data into its equivalent SAS variable format.
- “Effects of the SAS/ACCESS Interface on ADABAS Data” on page 97 explains how the SAS/ACCESS interface handles specific ADABAS data fields.

If you need help with SAS data sets and data libraries, their naming conventions, or any terms used in regard to the ACCESS procedure, refer to the *SAS Language Reference: Dictionary* and the *SAS Companion for z/OS*.

Case Sensitivity in the ACCESS Procedure

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

ACCESS Procedure Syntax for ADABAS

PROC ACCESS *<options>*;

Creating and Updating Statements

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

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

Database-Description Statements

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

NSS (LIBRARY | LIB= *library-identifier*

USER= *user-identifier*

PASSWORD | PW= *Natural-Security-password*);

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

PASSWORD | PW= *Adabas-password*

CIPHER | CC= *Adabas-cipher-code*

DBID= *Adabas-database-identifier*);

SYSFILE (NUMBER | NUM= *Adabas-system-file-number*

PASSWORD | PW= *Adabas-password*

CIPHER | CC= *Adabas-cipher-code*

DBID= *Adabas-database-identifier*);

SECFILE (NUMBER | NUM= *Natural-Security-system-file-number*

PASSWORD | PW= *Adabas-password*

CIPHER | CC= *Adabas-cipher-code*

DBID= *Adabas-database-identifier*);

Editing Statements

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

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

```

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

RUN;

```

Description

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

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

PROC ACCESS Statement Options

PROC ACCESS *options*;

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

You use the PROC ACCESS statement with database-description statements and certain procedure statements to create descriptors or SAS data files from DBMS data. See “Invoking the ACCESS Procedure” on page 60 for information about which procedure statements to use for each task.

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

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

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

DBMS=*ADABAS*

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

OUT=*<libref.>member-name*

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

VIEWDESC=*<libref.>view-descriptor*

specifies a view-descriptor that accesses the ADABAS data. VIEWDESC= is used only with the OUT= option. For example:

```
proc access dbms=adabas viewdesc=vlib.invg4
           out=dlib.invg4;
run;
```

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

CAUTION:

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

SAS Passwords for SAS/ACCESS Descriptors

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

The following table summarizes the levels of protection that SAS passwords have and their effects on access descriptors and view descriptors:

Table 5.1 Password and Descriptor Interaction

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

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

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

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

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

Assigning Passwords

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

Assigning Passwords with the DATASETS Procedure

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

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

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

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

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

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

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

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

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

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

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

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

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

Invoking the ACCESS Procedure

To invoke the ACCESS procedure you use the options described in “PROC ACCESS Statement Options” on page 58 and certain procedure statements. The options and statements that you choose are determined by your task.

- To create an access descriptor, you use the following syntax:

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

```
RUN;
```

- To create an access descriptor and a view descriptor in the same procedure, you use the following syntax:

```
PROC ACCESS DBMS=ADABAS;
```

```

CREATE libref.member-name.ACCESS;
    required database-description statements;
    optional editing statements;

```

```

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

```

```

RUN;

```

- To create a view descriptor from an existing access descriptor, you use the following syntax:

```

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

```

```

RUN;

```

- To update an access descriptor, you use the following syntax:

```

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

```

```

RUN;

```

- To update a view descriptor, you use the following syntax:

```

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

```

```

RUN;

```

ACCESS PROCEDURE Statements for ADABAS

ADBFILE Statement

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

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: DDM, SECURITY

Syntax

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

Details

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

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

Adabas-file number

is the ADABAS file number of the file to be accessed. The ADABAS file number is a number from 1 to the lower of 5,000 or the Association block size minus one. It is assigned when the ADABAS files are created with the ADABAS ADACMP utility.

Adabas-password

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

Adabas-cipher code

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

Adabas-database identifier

is the ADABAS database identifier (number) to be accessed. The database identifier is a numerical value from 1 to 65,535 that is assigned to each ADABAS database.

ASSIGN Statement

Indicates whether SAS variable names and formats are automatically generated.

Optional statement

Applies to: access descriptor

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

Default: NO

Syntax

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

Details

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

An editing statement, such as ASSIGN, must be specified after the CREATE and database-description statements when you create an access descriptor. See “CREATE Statement” on page 64 for more information.

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

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

Default SAS variable names are generated according to these rules:

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

If the same descriptor has another set of columns with duplicate names, the numeric suffix begins at the next highest number from the previous set of duplicate names. For example, if the descriptor has the duplicate names above and also has the DBMS names **customername**, **customernumber**, and **customernode**, the default SAS names would be **customer**, **custome1**, and **custome2**.

- If the column name contains characters that are not valid in SAS names (including national characters), SAS replaces these characters with underscores (_). For example, the column name **func\$** becomes the SAS variable name **func_**.

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

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

AN is the alias for the ASSIGN statement.

CONTENT Statement

Specifies a SAS date format or length.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN

Syntax

```
CONTENT column-identifier-1 <=> SAS-date-format | length | E
<... column-identifier-n <=> SAS-date-format | length | E>;
```

Details

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

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

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

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

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

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

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

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

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

Δ

CREATE Statement

Creates a SAS/ACCESS descriptor file.

Required statement

Applies to: access descriptor or view descriptor

Syntax

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

Details

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

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

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

Access descriptors

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

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

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

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

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

View descriptors

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

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

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

The order of the statements within this group usually does not matter; see the individual statement descriptions for any restrictions.

- 5 RUN statement: this statement is used to process the ACCESS procedure.

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

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

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

Examples

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

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

```

informat fo = date7.;
content fo = yymmdd8.;
mvf br occurs = 4
run;

```

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

```

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

```

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

```

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

  /* Create view descriptor */
  create vlib.empl204.view;
  select empid lastname hiredate salary dept
  sex birthdate;
  format empid 6.
    salary dollar12.2
    jobcode 5.
    hiredate datetime7.
    birthdate datetime7.;
  subset where jobcode=1204;

```

```
run;
```

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

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

DDM= Statement

Indicates the **NATURAL Data Definition Module (DDM)** name.

Optional statement

Applies to: access descriptor

Interacts with: NSS

Syntax

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

Details

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

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

If you delete or rename a SAS/ACCESS descriptor file, you do not delete or rename the descriptor file's underlying ADABAS file or NATURAL DDM. However, changing your DDM can affect your descriptor files. See "Effects of Changing an ADABAS File or NATURAL DDM on Descriptor Files" on page 107 for more information about how changing your DDM can affect your descriptor files.

DROP Statement

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

Optional statement

Applies to: access descriptor

Interacts with: RESET, SELECT

Syntax

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

Details

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

An editing statement, such as DROP, must follow the CREATE and database-description statements when you create an access descriptor. See “CREATE Statement” on page 64 for more information about the order of statements.

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

```
drop 3 5;
```

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

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

EXTEND Statement

Lists columns in the descriptor and gives information about them.

Optional statement

Applies to: access and view descriptors

Default ALL

Syntax

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

Details

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

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

If you use an editing statement, such as EXTEND, it must follow the CREATE statement and the database-description statements when you create a descriptor. See “CREATE Statement” on page 64 for more information about the order of statements.

You can specify EXTEND as many times as you want while creating a descriptor; specify EXTEND last in your PROC ACCESS code to see the completed descriptor

information. Or, if you are creating multiple descriptors, specify **EXTEND** before the next **CREATE** statement to list all the information about the descriptor you are creating.

The **EXTEND** statement can take one of the following arguments:

ALL

lists all the DBMS columns in the file, the positional equivalents, the two-character ADABAS names, the SAS variable informats, the database contents, occurrence ranges, descriptor types, and BY keys that are available for the access descriptor. When you are creating an access descriptor, ***NON-DISPLAY*** appears next to the column description for any column that has been dropped. When you are creating a view descriptor, ***SELECTED*** appears next to the column description for columns that you have selected for the view.

VIEW

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

column-identifier

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

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

```
extend 5;
```

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

```
extend 5 6 8;
```

FORMAT Statement

Changes a SAS format for a DBMS column.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: **ASSIGN**, **CONTENT**, **DROP**, **RESET**

Syntax

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

Details

The FORMAT statement changes a SAS variable format from its default format; the default SAS variable format is based on the data type of the DBMS column. (See “ACCESS Procedure Formats and Informats for ADABAS” on page 95 for information about the default formats that the ACCESS Procedure assigns to your DBMS data types.)

An editing statement, such as FORMAT, must follow the CREATE statement and the database-description statements when you create a descriptor. See “CREATE Statement” on page 64 for more information about the order of statements.

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

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

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

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

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

FMT is the alias for the FORMAT statement.

INFORMAT Statement

Changes a SAS informat for a DBMS column.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN, CONTENT, DROP, RESET

Syntax

```
INFORMAT column-identifier-1 <=> SAS-format-name  

     <...column-identifier-n <=> SAS-format-name>;
```

Details

The INFORMAT statement changes a SAS variable informat from its default informat; the default SAS variable informat is based on the data type of the DBMS column. (See “ACCESS Procedure Formats and Informats for ADABAS” on page 95 for information about the default informats that the ACCESS Procedure assigns to your DBMS data types.)

An editing statement, such as INFORMAT, must follow the CREATE statement and the database-description statements when you create a descriptor. See “CREATE Statement” on page 64 for more information about the order of statements.

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

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

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

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

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

INFMT is the alias for the INFORMAT statement.

KEY Statement

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

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN

Default blank

Syntax

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

Details

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

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

A data field is a good candidate for a BY key variable if it uniquely identifies a logical record. The incoming values of the data fields in a BY key variable are matched

to existing values in order to locate a position in which to insert new periodic groups. (A BY key variable is similar to a BY group or a BY variable in SAS.)

The KEY statement can have the following values:

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

An editing statement, such as KEY, must follow the CREATE statement and the database-description statements when you create a descriptor. See “CREATE Statement” on page 64 for more information about the order of statements.

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

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

LIST Statement

Lists columns in the descriptor and gives information about them.

Optional statement

Applies to: access descriptor or view descriptor

Default: ALL

Syntax

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

Details

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

If you use an editing statement, such as LIST, it must follow the CREATE statement and the database-description statements when you create a descriptor. See “CREATE Statement” on page 64 for more information about the order of statements.

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

The LIST statement can take one of the following arguments:

ALL

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

VIEW

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

column-identifier

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

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

```
list 5 8;
```

LISTINFO Statement

Shows additional data field information.

Optional statement

Applies to: access descriptor or view descriptor

Default: ALL

Syntax

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

Details

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

An editing statement, such as LISTINFO, must follow the CREATE statement and the database-description statements when you create a descriptor. See "CREATE Statement" on page 64 for more information about the order of statements.

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

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

The LISTINFO statement can take one of the following arguments:

ALL

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

VIEW

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

column-identifier

lists the specified DBMS columns, the ADABAS levels, ADABAS names, the lengths, the data types, the first-last character positions, and whether the column has been selected or dropped. If the column name contains special characters or national characters, enclose the name in quotation marks.

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

```
listinfo 5;
```

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

```
listinfo 5 6 8;
```

LISTOCC Statement

Lists occurrences for multiple value fields.

Optional statement

Applies to: access descriptor or view descriptor

Syntax

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

Details

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

If you use an editing statement, such as LISTOCC, it must follow the CREATE statement and the database-description statements when you create a descriptor. See “CREATE Statement” on page 64 for more information about the order of statements.

The LISTOCC statement takes the following argument:

column-identifier

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

```
listocc 5;
```

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

MVF Statement

Modifies the occurrences of a multiple-value field.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN

Syntax

MVF *column-identifier*

CONTENT *occurrence-1* <=> *E* | *SAS-date-format* | *length*
 <...*occurrence-n* <=> *E* | *SAS-date-format* | *length*>;

|
DROP *occurrence-1* <<TO> ... *occurrence-n*>;

|
FORMAT *occurrence-1* <=> *SAS-format-name*
 <... *occurrence-n* <=> *SAS-format-name*>;

|
INFORMAT *occurrence-1* <=> *SAS-format-name*
 <... *occurrence-n* <=> *SAS-format-name*>;

|
OCCURS<=> *number-of-occurrences*;

|
RENAME *occurrence-1* <=> *SAS-variable-name*
 < ... *occurrence-n* <=> *SAS-variable-name*>;

|
RESET *occurrence-1* <<TO> ... *occurrence-n*>;

|
SELECT *occurrence-1* <<TO> ... *occurrence-n*>;

Details

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

If you use an editing statement, such as MVF, it must follow the CREATE statement and the database-description statements when you create a descriptor. See “CREATE Statement” on page 64 for more information about the order of statements.

The MVF statement enables you to perform the following tasks:

- choose the number of occurrences by specifying a range of occurrences
- select individual occurrences or a range of occurrences
- drop individual occurrences or a range of occurrences
- reset individual occurrences or a range of occurrences
- change the format value for one or more occurrences
- change the informat value for one or more occurrences

- change the database content value for one or more occurrences
- rename the SAS variable name for one or more occurrences.

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

You can use the LISTOCC statement to review your changes.

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

The MVF statement can take one of the following subcommands:

CONTENT

enables you to change the DB content attribute of an individual occurrence. This subcommand can be used when creating access or view descriptors. Changing the DB content attribute of an occurrence has the same effect on the SAS formats and informats for ADABAS files and NATURAL DDMs as changing the DB content attribute of a column. See “CONTENT Statement” on page 63 for more information. For example, if the FIRSTORDERDATE column in the CUSTOMER DDM is a multiple-value field, and you want to change the DB content attribute for occurrences nine and ten, submit the following statement:

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

DROP

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

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

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

or

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

FORMAT

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

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

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

INFORMAT

enables you to change the informat attribute of an individual occurrence. This subcommand can be used when creating access or view descriptors. However, the

informat attribute cannot be changed in a view descriptor when you set ASSIGN=YES.

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

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

OCCURS

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

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

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

RENAME

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

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

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

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

RESET

enables you to reset the attributes of individual occurrences. This subcommand can be used when creating an access or view descriptor. Specifying the RESET subcommand for an occurrence has the same effect on occurrence attributes as specifying the RESET statement for a column. See "RESET Statement" on page 81 for more information.

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

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

or

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

SELECT

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

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

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

or

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

You can use the LISTOCC statement to review your changes.

NSS Statement

Specifies the NATURAL SECURITY options in the access descriptor.

Optional statement

Applies to: access descriptor

Interacts with: DDM and SECURITY

Syntax

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

Details

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

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

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

library-identifier

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

user-identifier

is an eight-character user identifier.

Natural-Security-password

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

QUIT Statement

Terminates the procedure.

Optional statement

Applies to: access descriptor or view descriptor

Syntax

QUIT;

Details

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

EXIT is the alias for the QUIT statement.

RENAME Statement

Modifies the SAS variable name.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN, RESET

Syntax

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

Details

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

An editing statement, such as RENAME, must follow the CREATE statement and the database-description statements when you create a descriptor. See “CREATE Statement” on page 64 for more information about the order of statements.

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

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

When creating a view descriptor that is based on this access descriptor, you can specify the RESET statement or another RENAME statement to rename the variable again, but the new name applies only in that view. When you create other view descriptors, the SAS variable names are derived from the access descriptor.

- If you specify the **YES** value in the ASSIGN statement, you can use the RENAME statement to change SAS variable names only while creating a specific access descriptor. As described earlier in the ASSIGN statement, SAS variable names that are saved in an access descriptor are always used when creating view descriptors that are based on it.

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

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

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

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

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

RESET Statement

Resets DBMS columns to their default settings.

Optional statement

Applies to: access descriptor or view descriptor

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

Syntax

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

Details

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

If you use an editing statement, such as **RESET**, it must follow the **CREATE** statement and the database-description statements when you create a descriptor. See “**CREATE Statement**” on page 64 for more information about the order of statements.

Access descriptors

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

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

View descriptors

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

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

The **RESET** statement can take one of the following arguments:

ALL

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

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

column-identifier

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

```
reset 3;
```

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

SECFILE Statement

Specifies parameters for the NATURAL SECURITY system file.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: SECURITY

Syntax

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

Details

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

If you specified SECURITY=YES in the access descriptor, you cannot change the values for the password and the cipher code in the view descriptor based on this access descriptor. However, if no values were specified in the parent access descriptor, then the values can be entered in the view descriptor, even when the SECURITY=YES statement has been issued.

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

Natural-Security-file-number

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

Adabas-password

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

Adabas-cipher code

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

Adabas-database-identifier

is the ADABAS database identifier (number) to be accessed. The database identifier is a numerical value from 1 to 65,535 that is assigned to each ADABAS database.

SECURITY Statement

Controls the enforcement of security specifications.

Optional statement

Applies to: access descriptor

Interacts with: ADBFILE, SECFILE, SYSFILE

Default: NO

Syntax

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

Details

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

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

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

SELECT Statement

Selects DBMS columns for the view descriptor.

Required statement

Applies to: view descriptor

Interacts with: RESET

Syntax

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

Details

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

If you use an editing statement, such as SELECT, it must follow the CREATE statement when you create a view descriptor. See “CREATE Statement” on page 64 for more information about the order of statements.

The SELECT statement can take one of the following arguments:

ALL

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

column-identifier

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

```
select 1 2 3;
```

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

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

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

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

SUBSET Statement

Adds or modifies selection criteria for a view descriptor.

Optional statement

Applies to: view descriptor

Syntax

SUBSET <selection-criteria>;

Details

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

An editing statement, such as SUBSET, must follow the CREATE statement when you create a view descriptor. See "CREATE Statement" on page 64 for more information about the order of statements.

The selection-criteria argument can be either a WHERE clause or a SORT clause. For more information about the WHERE clause, see "WHERE Clause in an ADABAS View Descriptor" on page 88. For more information about the SORT clause, see "SORT Clause in a View Descriptor" on page 93. You can use either SAS variable names or DBMS column names, in your selection criteria. Specify your WHERE clause and SORT clause by using separate SUBSET statements. For example, you can submit the following SUBSET statements:

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

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

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

SYSFILE Statement

Specifies parameters for the system file containing DDMs.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: SECURITY

Syntax

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

Details

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

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

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

Adabas-system-file-number

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

Adabas-password

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

Adabas-cipher code

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

Adabas-database-identifier

is the ADABAS database identifier (number) to be accessed. The database identifier is a numerical value from 1 to 65,535 that is assigned to each ADABAS database.

UPDATE Statement

Updates a SAS/ACCESS descriptor file.

Optional statement

Applies to: access descriptor or view descriptor

Syntax

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

Details

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

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

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

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

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

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

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

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

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

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

```

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

```

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

```

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

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

```

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

```

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

```

WHERE Clause in an ADABAS View Descriptor

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

View WHERE Clause Syntax

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

```

field-name<(occurrence)> operator value
field-name<(occurrence)> range-operator

```

low-value * high-value

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

field-name

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

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

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

(occurrence)

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

operator

can be one of the following comparison and logical operators:

= or EQ	equal to
> or GT	greater than
< or LT	less than
!= or \neq or NE	not equal to
\geq or GE or GTE	greater than or equal to
\leq or LE or LTE	less than or equal to

range-operator

can be one of the following operators:

= or EQ or SPANS	within the range (inclusive)
------------------	------------------------------

value or high-value or low-value

is a valid value for the data field.

View WHERE Clause Examples

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

Specifying Conditions with the SPANS Operator

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

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

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

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

Specifying Expressions

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

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

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

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

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

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

Specifying Values in Character Fields

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

```
subset where lastname = Smith
```

The next example extracts data for Smith:

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

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

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

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

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

You can also use double quotation marks, for example,

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

Specifying Numeric Format Values

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

```
subset where horsepower = 2.5
```

Specifying Dates

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

However, the ACCESS procedure provides you the ability to specify a SAS date format with the CONTENT statement. Then, numeric values are converted to SAS dates. To reference them in a view WHERE clause, use informat representation (without the 'D' at the end as in SAS). See "CONTENT Statement" on page 63 for more information about specifying a SAS date format with the CONTENT statement.

Specifying Values in Superdescriptor Fields

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

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

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

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

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

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

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

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

```
subset where state_zi = E3E7F0F1
```

The comparable SAS WHERE clause would be

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

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

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

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

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

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

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

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

A comparable SAS WHERE clause would be

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

Specifying Values in Subdescriptor Fields

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

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

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

The data field that ZIPLAST is based on is defined as

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

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

View WHERE clause:

```
subset where ziplast2 = 01
```

SAS WHERE clause:

```
where ziplast2 = 01
```

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

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

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

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

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

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

The comparable SAS WHERE clause would be

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

Specifying Values in Multiple-Value Fields

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

```
subset where card eq visa
```

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

For more information about and examples of using multiple-value fields in selection criteria, see “Using Multiple-Value Fields in Selection Criteria” on page 122.

Specifying Values in Periodic Group Fields

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

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

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

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

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

For more information about and examples of using periodic group fields in selection criteria, see “Using Multiple-Value Fields in Selection Criteria” on page 122.

SORT Clause in a View Descriptor

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

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

View SORT Clause Syntax

The syntax for the SORT clause is

```
SUBSET SORT field-name <,field-name> <,field-name> <option>
```

The elements of the SORT clause are described below.

field-name

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

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

option

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

<ASCENDING | ASCENDISN | DESCENDING>

ASCENDING

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

ASCENDISN

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

DESCENDING

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

SORT Clause Examples

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

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

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

```
subset sort name descending
```

Creating and Using ADABAS View Descriptors Efficiently

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

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

When ADABAS cannot use an inverted list, it sequentially scans the entire file. You cannot guarantee that ADABAS will use an inverted list to process a condition on a descriptor data field, but you can write selection criteria that enable ADABAS to use available inverted lists effectively.

- Select only the data fields your program needs. Selecting unnecessary data fields adds extra processing time and requires more memory.
- Use a BY statement to specify the order in which logical records are presented to SAS only if SAS needs the data in a particular order for subsequent processing. You can use ADABAS descriptor data fields only.

As an alternative to using a BY statement, which consumes CPU time each time you access the ADABAS file, you could use the SORT procedure with the OUT= option to create a sorted SAS data file. In this case, SAS, not ADABAS, does the sorting. This is a better approach for data that you want to use many times.

- If a view descriptor describes a large amount of ADABAS data and you will use the view descriptor often, it might be more efficient to extract the data and place it in a SAS data file. See “Performance Considerations” on page 34 for more information about when it is best to extract data.
- If you don’t need all occurrences of multiple-value fields, limit the number of occurrences with the MVF statement.
- If you reference data fields in selection criteria that are not ADABAS descriptors, it is generally more efficient to put those conditions in a SAS WHERE clause, not in the view descriptor WHERE clause.
- To optimize WHERE clause processing, the ADABAS interface view engine uses the ADABAS L3 command when possible. However, a number of restrictions must be satisfied before the L3 command can be used. For these restrictions, see “How the SAS/ACCESS Interface to ADABAS Works” on page 102.

ACCESS Procedure Formats and Informats for ADABAS

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

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

- The ADABAS interface view engine uses ADABAS standard length and type for reading and updating ADABAS data (except for variable-length fields and DB Content overrides). NATURAL DDMs have no effect other than to use DDM length and decimals to set SAS formats.
- Length and decimal points specified by DDMs might conflict with the ADABAS file definition (for example, not big enough, too big, and so on). If so, the ADABAS standard length is used to set default SAS formats.
- Packed, unpacked, and binary types can hold very large numeric data values. SAS can maintain precision up to sixteen digits. Unpacked fields larger than sixteen bytes are converted to the character hexadecimal type upon which no numeric operations can occur. Therefore, precision is not a problem. For large packed and binary fields, however, you must be aware that precision can be lost when data values exceed sixteen digits.
- If the standard length is 0 (that is, if the data field has a variable length), the ACCESS procedure chooses a default length.
 - The default length for an alphanumeric is 20.
 - The default length for a numeric is the maximum length before assuming a character hexadecimal type. Packed is 15 bytes (29 digits and a sign),

unpacked is 16 bytes (16 digits and a sign), binary is 8 bytes, fixed is 4 bytes, and float is 8 bytes.

- Superdescriptors and subfields are given an ADABAS type of character unless all of the parent fields are numeric. Then, they are given an ADABAS type of binary. Their length is calculated by totaling the number of bytes in the individual parent's from-to specification. If the length of a binary superdescriptor or subdescriptor is greater than 8, the SAS format is changed from numeric to character hexadecimal.
- Subdescriptors and subfields take the type of their parent and the length of their from-to specification.
- Phonetic descriptors are alphanumeric and use the length of the phonetic parent. Any retrieval of a phonetic descriptor is actually retrieval of its parent.
- If ADABAS data falls outside the valid SAS data ranges, you will get an error message in the SAS log when you try to read the data. For example, an ADABAS date might not fall in the valid SAS date range.

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

Table 5.2 SAS Formats and Informats for ADABAS Data Types in an ADABAS File

ADABAS Type	Description	Standard Length in Bytes	SAS Format and Informat
A	alphanumeric	≤ 200	$\$ADBLEN$.
		> 200	$\$200$
B	binary (unsigned)	≤ 4	$(2 \times ADBLEN) + 1$
		> 4 and ≤ 8	$(2 \times ADBLEN)$.
		> 8 and ≤ 100	$\$HEX(2 \times ADBLEN)$.
		> 100	$\$HEX200$.
F	fixed (signed)		8.
G	floating point (signed)		BEST12.
P	packed decimal (signed)		$(2 \times ADBLEN + 1)$.
U	unpacked decimal	≤ 16	$(ADBLEN + 1)$.
	(zoned decimal) (signed)	> 16	$\$HEX(2 \times ADBLEN)$.

The following information applies to this table:

- $ADBLEN$ = ADABAS standard length (in bytes). If the standard length equals 0, then the interface view engine sets the length based on the data type, as follows: A=20, B=8, F=4, G=8, P=15, and U=16.
- Binary data that is
 - ≤ 4 bytes is treated as signed numbers
 - ≤ 8 bytes and > 4 bytes is treated as positive (unsigned) numbers
 - > 8 bytes is treated as character hexadecimal data.

- Numeric values greater than 16 displayable digits can lose precision.

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

Table 5.3 SAS Formats and Informats for ADABAS Data Types in a NATURAL DDM

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

The following information applies to this table:

- *DDMLEN* = DDM digits to the left of the decimal point.
- *DDMDEC* = DDM digits to the right of the decimal point.
- *ADBLEN* = ADABAS standard length in bytes. If the standard length equals 0, then the interface view engine sets the length based on the data type, as follows: A=20, B=8, F=4, G=8, P=15, and U=16.
- *DECPT* = 1 when DDM digits to the right of the decimal point are greater than 0.
- *DECPT* = 0 when DDM digits to the right of decimal point are equal to 0.
- *SIGNPT* = 1 when numeric type is signed data (fixed, float, packed, unpacked, and binary ≤ 4).
- *SIGNPT* = 0 when numeric type is unsigned data (binary > 4 and ≤ 18).
- Binary data that is
 - ≤ 4 bytes is treated as signed numbers
 - ≤ 8 bytes and > 4 bytes is treated as positive (unsigned) numbers
 - > 8 bytes is treated as character hexadecimal data.
- Numeric values greater than 16 displayable digits can lose precision.

Effects of the SAS/ACCESS Interface on ADABAS Data

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

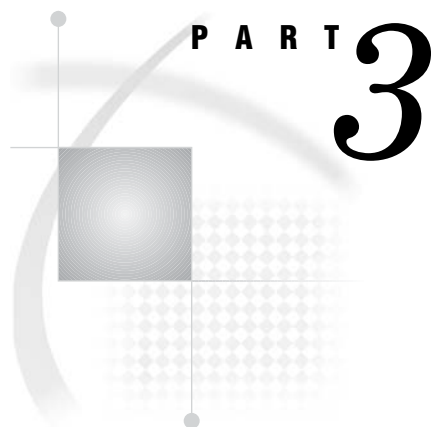
- Multiple-value field occurrences are mapped to multiple SAS variables. For example, if the ADABAS data has a multiple-value field named `JOBTITLE` with two occurrences, the resulting SAS variables would be `JOBTITL1` and `JOBTITL2`.
- Periodic group occurrences are mapped to multiple SAS observations.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

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

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

If you use a variable for a phonetic descriptor in a SAS `WHERE` clause, the interface view engine must be able to process the entire SAS `WHERE` clause.

- For a multiple-value data field, the ACCESS procedure creates SAS variables that reference individual occurrences and a SAS variable that references all occurrences to perform special `WHERE` clause queries. For example, in the NATURAL DDM named `CUSTOMERS`, the `BRANCH-OFFICE` data field is a multiple-value data field with four occurrences. The ACCESS procedure creates SAS variables named `BRANCH_1`, `BRANCH_2`, and so on, and a SAS variable named `BR_ANY`. For more information and examples, see “Using Multiple-Value Fields in Selection Criteria” on page 122.
- For a periodic group data field, the ACCESS procedure creates a SAS variable for the occurrence number within the periodic group. For example, in the NATURAL DDM named `CUSTOMERS`, the `SIGNATURE-LIST` data field is a periodic group for data fields `LIMIT` and `SIGNATURE`. PROC ACCESS creates a SAS variable named `SL_OCCUR` for the occurrence numbers. For more information and examples, see “Periodic Group Fields in Selection Criteria” on page 124.



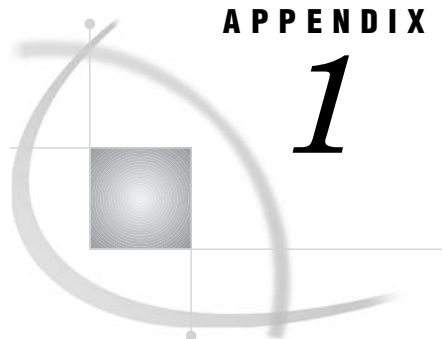
Appendices

Appendix 1 **Information for the Database Administrator** 101

Appendix 2 **Advanced Topics** 115

Appendix 3 **Example Data** 131

Appendix 4 **Recommended Reading** 155



APPENDIX

1

Information for the Database Administrator

<i>Introduction to the Information for the Database Administrator</i>	101
<i>How the SAS/ACCESS Interface to ADABAS Works</i>	102
<i>How the ADABAS Interface View Engine Works</i>	102
<i>Calls Made on Behalf of the ACCESS Procedure</i>	103
<i>Calls Made by Other SAS Procedures</i>	103
<i>Retrieval Processing</i>	103
<i>Retrievals with No WHERE Clause and No Sorting Criteria</i>	104
<i>Retrievals with Only a WHERE Clause</i>	105
<i>Retrievals with Sorting Criteria</i>	106
<i>Update Processing</i>	106
<i>Competitive Updating and Logical Transaction Recovery</i>	107
<i>Effects of Changing an ADABAS File or NATURAL DDM on Descriptor Files</i>	107
<i>Changes That Have No Effect on Existing View Descriptors</i>	107
<i>Changes That Might Have an Effect on Existing View Descriptors</i>	108
<i>Changes That Cause Existing View Descriptors to Fail</i>	108
<i>Data Security with ADABAS</i>	108
<i>How the Interface View Engine Uses Security Specifications</i>	108
<i>SAS Security</i>	110
<i>ADBSE User Exit</i>	110
<i>Effects of Changing Security Options</i>	110
<i>Controlling Data Locks with ADABAS</i>	111
<i>Maximizing ADABAS Performance</i>	111
<i>Debug Information for ADABAS</i>	112
<i>System Options for PROC ACCESS and the Interface View Engine</i>	112
<i>ADBAUSE System Options Default Values</i>	112
<i>View Engine ADBEUSE System Options Default Values</i>	113

Introduction to the Information for the Database Administrator

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 system options.

How the SAS/ACCESS Interface to ADABAS Works

When you use the ACCESS procedure to create a SAS/ACCESS access descriptor file, SAS calls ADABAS to get a description of the ADABAS data. When you create a view descriptor file, SAS 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 is in an ADABAS file, the SAS 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, SAS 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 SAS 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 it 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. SAS 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

The SAS/ACCESS interface view engine uses the ADABAS multifetch feature when reading data records from ADABAS. When data records are requested from ADABAS, this feature buffers multiple data records and transfers the records to the engine. The data records are returned to the engine in the ADABAS record buffer with information

about each record that is returned in the ISN buffer. Therefore, the number of records that are returned depends on the sizes of the record buffer and ISN buffer.

The read commands that use the multifetch feature in the engine are L1, L2, L3, and L9 commands (the engine does not use the feature when issuing L4 commands). In cases where there is a large number of periodic group occurrences selected in the view descriptor, the multifetch feature might not be used to read the data records from ADABAS.

The type of processing and the subset of ADABAS commands used by the interface view engine depends on the following conditions:

- 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):

Table A1.1 ADABAS Field Formats and Corresponding Default WHERE Clauses

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 or 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 might also be issued for an ADABAS descriptor that has the NULL SUPPRESS option. That is, ADABAS records might exist that are not pointed to by the ISN list. Δ

Retrievals with Only a WHERE Clause

If you specify a WHERE clause (either view or SAS), the engine typically issues S1, S8, and L1 or 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 127. Δ

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 or L4 commands.

Note: In SAS 9.1 and later, the ADABAS engine can issue an L1 command to ADABAS when an ISN is specified in a SAS WHERE clause. With this method, only one record is read instead of the complete table, resulting in a performance enhancement. Δ

The L1 command is issued if an ISN is specified in a SAS WHERE clause and all of the following conditions are met:

- the WHERE clause must be a SAS WHERE clause
- no view descriptor SUBSET can be used
- the WHERE clause can contain only a single condition
- the operator must be EQ or =
- sorting criteria cannot be specified
- option ADBL3 must be set to NO (its default value).

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 or 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 ADABAS Observation” on page 118. Deleting records when the view descriptor or ADABAS data contains periodic group fields is discussed in “Deleting an ADABAS Observation” on page 118.

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 can 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 PROC ACCESS, 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 SAS 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 with ADABAS

SAS 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 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 might need to be provided for each of three ADABAS files that the interface view engine might reference during the execution of a single SAS procedure. The following are the three ADABAS files:

- 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 following are the four fields of information associated with each ADABAS file:

- 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 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 might 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.

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

(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 and update authorization for a library identifier and user identifier. If they do not have the appropriate authorization, an error message is displayed.

SAS Security

To secure data from accidental update or deletion, you can do the following on SAS 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.
- 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 SAS. See “View Engine ADBEUSE System Options Default Values” on page 113.

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 with ADABAS

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 about how the interface view engine handles locking, see "Competitive Updating and Logical Transaction Recovery" on page 107.

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 enable ADABAS to use ADABAS descriptor data fields efficiently? See "Creating and Using ADABAS View Descriptors Efficiently" on page 94 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 is restricted for the program being used.
- Is the data going to be used by more than one procedure in a single SAS session? If so, consider extracting the data and placing it 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 34 for circumstances when extracting data is the more efficient method.
- Does 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 with ADABAS" on page 111.

- 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 ADABAS

If you are experiencing a problem with the SAS/ACCESS interface to ADABAS, the Technical Support staff at SAS might ask you to provide additional debug information. They might 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.

System Options for PROC ACCESS 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 system options. The two CSECTs are: ADBAUSE for the ACCESS procedure and ADBEUSE for the interface view engine.

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

ADBAUSE System Options Default Values

The following system 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.

Table A1.2 ADBAUSE System Options Using 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.

Option	Default	Purpose
ADBSYSCC	blanks	System file cipher code.
ADBSYSDB	0	System file database identifier.
ADBSYSFL	15	System file number.
ADBSYSPW	blanks	System file password.

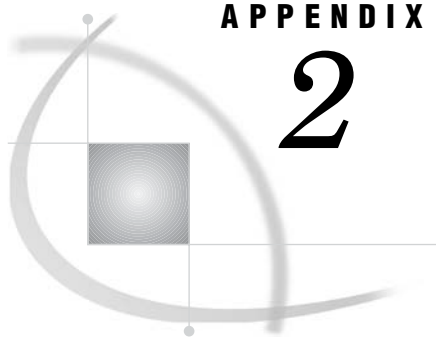
View Engine ADBEUSE System Options Default Values

The system options shown in the following table 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 A1.3 ADBEUSE System 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.
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.

Option	Default	Purpose
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.
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.



APPENDIX

2

Advanced Topics

<i>Introduction to Advanced Topics</i>	115
<i>Data Set Options for ADABAS</i>	116
<i>Using Multiple View Descriptors</i>	118
<i>Deleting an ADABAS Observation</i>	118
<i>Adding an ADABAS Observation</i>	118
<i>Using a BY Key To Resolve Ambiguous Inserts</i>	119
<i>BY Key Examples</i>	119
<i>By Key Example 1</i>	120
<i>By Key Example 2</i>	120
<i>By Key Example 3</i>	120
<i>BY Key Considerations</i>	121
<i>Missing Values (Nulls)</i>	121
<i>Using Multiple-Value Fields in Selection Criteria</i>	122
<i>WHERE Clause Examples</i>	123
<i>WHERE Clause Example 1</i>	123
<i>WHERE Clause Example 2</i>	123
<i>WHERE Clause Example 3</i>	124
<i>Periodic Group Fields in Selection Criteria</i>	124
<i>WHERE Clause Examples</i>	125
<i>WHERE Clause Example 1</i>	125
<i>WHERE Clause Example 2</i>	126
<i>WHERE Clause Example 3</i>	126
<i>Using a SAS WHERE Clause for Selection Criteria</i>	127
<i>SAS WHERE Clause Conditions Acceptable to ADABAS</i>	128
<i>SAS WHERE Clause Conditions Not Acceptable to ADABAS</i>	129
<i>When a SAS WHERE Clause Must Reference Descriptor Data Fields</i>	129
<i>Deciding How to Specify Selection Criteria</i>	130
<i>View WHERE Clause</i>	130
<i>SAS WHERE Clause</i>	130

Introduction to Advanced Topics

This appendix contains details about some advanced topics such as using data set options, using multiple view descriptors, deleting and adding observations, using BY keys, and processing null values, as well as topics pertaining to selection criteria. The discussions supplement other portions of this document.

Data Set Options for ADABAS

In order for the ADABAS interface view engine to obtain ADABAS dictionary information, it needs certain ADABAS information. Specifically, the engine needs either a NATURAL DDM name or an ADABAS file number, in addition to a library identifier, a user identifier, passwords, cipher codes, and a database identifier.

If any of this information is required to access an ADABAS file or a NATURAL DDM but is not specified in the SAS/ACCESS view descriptor or cannot be obtained from either the ADBEUSE or ADBAUSE CSECT, you must use the appropriate data set option in your SAS procedure statement to supply the appropriate value.

Data set options enable you to specify these values. Data set options also enable you to override certain values that are specified in view descriptors but not enforced by ASSIGN SECURITY=YES.

Each data set option is an option in the DATA= specification where DATA= specifies a view descriptor that will be used as input to a SAS procedure. Data set options apply only for the duration of that procedure.

The following example executes the FSEDIT procedure using a view descriptor named VLIB.USAINV. The data set option specified in the PROC statement will execute ADABAS using the NATURAL SECURITY password INVOICE.

```
proc fsedit data=vlib.usainv (adbnatpw='invoice');
run;
```

The available data set options appear below. Options marked with an asterisk (*) are enforced by ASSIGN SECURITY=YES. That is, if ASSIGN SECURITY=YES, the values specified in the view descriptor take precedence over values specified with a data set option; the data set option is ignored.

ADBCC=*'cipher-code'*

specifies a cipher code for the target ADABAS file.

ADBDBID=*database-identifier*

specifies a database identifier for the target ADABAS file.

ADBDEL=N|NO|Y|YES

enables you to override the default value for the interface view engine's system option that determines whether a record containing periodic group fields should be completely deleted or its periodic group fields set to nulls. The default is set by the ADBDEL systems option in the ADBEUSE CSECT.

NO means set the fields to null; YES means delete the entire record.

ADBDDM=*'ddm-name'*

specifies a NATURAL Data Definition Module (DDM) name. The ADBFILE and ADBDDM data set options are mutually exclusive. If you specified a DDM name in the view descriptor, you can use ADBDDM, but you cannot use ADBFILE. If you specified an ADABAS file number instead, you can use ADBFILE but not ADBDDM.

ADBFILE=*file-number*

specifies an ADABAS file number. The ADBFILE and ADBDDM data set options are mutually exclusive. If you specified a DDM name in the view descriptor, you can use ADBDDM, but you cannot use ADBFILE. If you specified an ADABAS file number instead, you can use ADBFILE but not ADBDDM.

ADBFMTL=*length*

specifies the length for the ADABAS format buffer. The minimum value is 100. The default value is 500.

ADBISNL=*length*

specifies the length for the ADABAS ISN buffer. The minimum value is 100. The default value is 5,000.

ADBL3=N|NO|Y|YES|O|ONLY

controls the use of the ADABAS L3 command by the interface view engine and what commands are used when L3 cannot be used. The L3 command optimizes WHERE and sort processing, with dramatic results for very large ADABAS files. However, there are limitations on when the command can be used. See “Retrievals with Only a WHERE Clause” on page 105 for more information.

NO means the L3 command should not be used; YES means L3 is used and S1 and S9 are used if L3 cannot be used; ONLY means L3 is used and S2 is used, or an error is generated, when L3 cannot be used.

ADBNATAP=*'library-id'* *

specifies a NATURAL SECURITY library identifier.

ADBNATPW=*'password'* *

specifies a NATURAL SECURITY user password.

ADBNATUS=*'user-id'* *

specifies a NATURAL SECURITY user identifier.

ADBPW=*'password'* *

specifies an ADABAS password for the target ADABAS file.

ADBRECL=*length*

specifies the length for the ADABAS record buffer. Acceptable values are in the range of 2,100–32,767. The default value is 7,500.

ADBSCHL=*length*

specifies the length for the ADABAS search buffer. The minimum value is 100. The default value is 500.

ADBSECCC=*'cipher-code'* *

specifies an ADABAS cipher code for the NATURAL SECURITY system file.

ADBSECDB=*database-identifier*

specifies an ADABAS database identifier for the NATURAL SECURITY system file.

ADBSECFL=*file-number*

specifies an ADABAS file number for the NATURAL SECURITY system file.

ADBSECPW=*'password'* *

specifies an ADABAS password for the NATURAL SECURITY system file.

ADBSYSCC=*'cipher-code'* *

specifies an ADABAS cipher code for the DDM system file.

ADBSYSDB=*database-identifier*

specifies an ADABAS database identifier for the DDM system file.

ADBSYSFL=*file-number*

specifies an ADABAS file number for the DDM system file.

ADBSYSPW=*'password'* *

specifies an ADABAS password for the DDM system file.

ADBTRACE=*option*

specifies a trace option, which analyzes problems in SAS software. The default is ADBTRACE=0. If you specify ADBTRACE=1, WHERE clauses are displayed in

the log. For more information about ADBTRACE, see “Debug Information for ADABAS” on page 112.

ADBVALL=length

specifies the length for the ADABAS value buffer. Acceptable values are in the range of 100–32,767. The default value is 300.

Using Multiple View Descriptors

You can use more than one view descriptor in a single SAS session, but only one can be open for updating. This is the default mode of operation.

For information about how to modify the engine to support multiple view descriptors in a single SAS session, contact your SAS Technical Support Representative.

Deleting an ADABAS Observation

If the ADABAS file on which a view descriptor is based does not contain a periodic group, deleting an observation (for example, with the FSEEDIT procedure DELETE command) causes a logical record to be deleted from the ADABAS data.

If the ADABAS file on which a view descriptor is based does contain a periodic group (the periodic group may or may not be included in the view descriptor), the results of deleting an observation depend on the status of the ADBDEL systems option, which is set either in the ADBEUSE CSECT (see “System Options for PROC ACCESS and the Interface View Engine” on page 112) or by a data set option (see “Data Set Options for ADABAS” on page 116).

- When ADBDEL=N (which is the default setting), the following results occur:
 - If there is only one periodic group occurrence (regardless of how many periodic group fields are in the view descriptor) and there are no other periodic group fields in the ADABAS file, deleting the observation containing the one occurrence causes the logical record containing the occurrence to be deleted.
 - If there are multiple occurrences for any periodic group field(s) in the view descriptor or if there are other periodic group fields in the ADABAS file, deleting the observation containing values from a periodic group occurrence causes the selected values for that occurrence to be set to null. The record is not deleted.
- If ADBDEL=P, the entire logical record is deleted, even if there are multiple occurrences for periodic group fields in the view descriptor or if there are other periodic group fields in the ADABAS file.

Adding an ADABAS Observation

Adding ADABAS data as a result of update operations from various SAS procedures might cause the interface view engine to decide whether to add a new logical record to the ADABAS file or modify an existing logical record, for example, add an occurrence to a periodic group. The purpose of the engine making this determination is to reduce data redundancy.

The engine compares values in the new observation to be added to values in the previous observation. If the contents of the previous observation do not help determine whether to add or modify, a new logical record is added.

However, some of the new values might already reside in the ADABAS file, so a new record is not necessary. This situation occurs if a periodic group is included in a view descriptor, and the new data (which does not reside in the ADABAS file) occurs only in variables corresponding to data fields that are part of that periodic group.

The interface view engine can determine whether this situation exists. If not, a new logical record can be added. If so, an existing record can be modified. The optional BY key specification makes this possible. See “Using a BY Key To Resolve Ambiguous Inserts” on page 119.

Using a BY Key To Resolve Ambiguous Inserts

When the interface view engine is called to examine additional ADABAS records in order to add a new periodic group occurrence, the engine must decide whether to add a new logical record or modify an existing one. The purpose is to reduce data redundancy.

You can help in the resolution of this decision by specifying a BY key. You can specify BY keys in the access descriptor by using the KEY statement. If ASSIGN NAMES=NO, you can use the KEY statement to specify BY keys in the view descriptor. Only elementary data fields that are designated as ADABAS descriptors can be specified as BY keys.

A BY key is a set of match variables. A data field is a good candidate for a BY key if it uniquely identifies a logical record.

A BY key is similar to a BY group in SAS, which groups observations based on one or more fields. Many SAS procedures process records in BY groups. Also, some updates in the DATA step are performed by matching specified BY variables in different data sets. A similar matching process occurs with BY key data fields in the SAS/ACCESS interface to ADABAS.

The BY key comparison process is as follows:

- 1 If values for a BY key match a record already in the ADABAS file, it will be modified. That is, the interface view engine inserts a new occurrence within a periodic group.
- 2 If values for a BY key do not match an existing record, a new record is added to the ADABAS file.

BY Key Examples

The following examples illustrate that using a BY key helps keep data organized and prevents unnecessary duplication of data.

Suppose you are working with the following two ADABAS logical records, which make up three SAS observations as shown in the following output. The data field named DF1 is specified as a BY key. DF2 is a periodic group consisting of data fields DF21 and DF22.

Output A2.1 By Key Example Containing Two ADABAS Logical Records of Three SAS Observations

Data Fields	DF1	DF2		
		DF21	DF22	
Record 1	A	CCC	1	(obs 1)
		CCC	2	(obs 2)
Record 2	B	DDD	3	(obs 3)

By Key Example 1

You are in the FSEDIT procedure on observation 1. You enter an ADD or a DUP command and the values A, CCC, and 4. This is not an ambiguous insert, and a BY key is not required. The following output shows the result.

Output A2.2 Results of Entering an ADD or DUP Command

Data Fields	DF1	DF2		
		DF21	DF22	
Record 1	A	CCC	1	(obs 1)
		CCC	2	(obs 2)
		CCC	4	(new observation (obs 4))
Record 2	B	DDD	3	(obs 3)

By Key Example 2

You are in the FSEDIT procedure on observation 1. You enter an ADD or a DUP command and the values B, DDD, and 5 for data fields DF1, DF21, and DF22, respectively. This is an ambiguous insert because all the values you are entering are different than the ones in observation 1. If there were not a BY key, the result would be as shown in the following output.

Output A2.3 Results of an Ambiguous Insert

Data Fields	DF1	DF2		
		DF21	DF22	
Record 1	A	CCC	1	(obs 1)
		CCC	2	(obs 2)
		CCC	4	(obs 3)
Record 2	B	DDD	3	(obs 4)
Record 3	B	DDD	5	(new observation)

With a BY key, the engine locates the BY key value DF1=B. The following output shows the result.

Output A2.4 Results with a BY Key

Data Fields	DF1	DF2		
		DF21	DF22	
Record 1	A	CCC	1	(obs 1)
		CCC	2	(obs 2)
		CCC	4	(obs 3)
Record 2	B	DDD	3	(obs 4)
		DDD	5	(new observation)

By Key Example 3

You are in the FSVIEW procedure, looking at the first three observations. You decide to add the values B, DDD, and 7 at the end. The current position is the third observation on the display. The following output shows the result with no BY key.

Output A2.5 Results without a By Key

Data Fields	DF1	DF2		
		DF21	DF22	
Record 1	A	CCC	1	(obs 1)
		CCC	2	(obs 2)
		CCC	4	(obs 3)
Record 2	B	DDD	3	(obs 4)
		DDD	5	(obs 5)
Record 3	B	DDD	7	(new observation)

The following output shows the result with a BY key.

Output A2.6 Results with a BY Key

Data Fields	DF1	DF2		
		DF21	DF22	
Record 1	A	CCC	1	(obs 1)
		CCC	2	(obs 2)
		CCC	4	(obs 3)
Record 2	B	DDD	3	(obs 4)
		DDD	5	(obs 5)
		DDD	7	(new observation)

BY Key Considerations

When specifying BY keys for your view descriptors, keep in mind the following considerations:

- A duplicate consecutive observation results in an additional occurrence in any periodic group in the view descriptor.
- If you do an insert from an observation that has all missing values, the interface view engine inserts a record that is equivalent to all zeros and blanks.
- The APPEND function of the SAS Component Language (SCL) must be preceded by a call to the SET function. Otherwise, APPEND inserts an observation that is equivalent to all zeros and blanks because the insert is too ambiguous for the interface view engine to resolve.
- If a view descriptor includes a periodic group and you try to add an observation that is another occurrence in that periodic group, the add might fail if you are attempting to add more occurrences than the periodic group field definition allows. One of the following will occur, depending on whether a BY key is specified:
 - If no BY key is defined, and
 - if the last observation was not created from the periodic group, a new logical record is added.
 - if the last observation was created from the periodic group, the add fails with a return code, and a new record is then added.
 - If a BY key is defined and the periodic group is selected to have an added occurrence, the add fails and a message displays.

Missing Values (Nulls)

When the interface view engine is reading ADABAS data and constructing an observation, it could find missing (null) values for data fields within an observation.

The interface view engine uses the L1, L2, L3, and L4 commands to retrieve ADABAS data. The values are returned in the record buffer using the standard length and format defined for that field. (Standard length is not used if you have specified a value for the DB Content field or the field is a variable length field.) If the field's value is null, the data is returned in the format in effect for that field.

Formats and their corresponding null values are listed below.

Table A2.1 ADABAS Data Formats and Null Values

Format	Null Value
Alphanumeric	blanks
Binary	binary zeros
Fixed Point	binary zeros
Unpacked Decimal	unpacked decimal zeros
Packed Decimal	packed decimal zeros
Floating point	binary zeros

When an ADABAS record is read, the interface view engine is unable to tell whether a field has a value of zero (for numeric fields) or blanks (for alphanumeric fields) or truly has a null value. This is also true when you are updating. When you are using the FSEDIT procedure, if a value of zero or missing is used to modify an existing record, zeros are placed in the ADABAS record buffer and subsequently added to the ADABAS file. Blanks are placed in the record buffer if a blank or missing value was supplied for an alphanumeric field.

Since SAS missing values are stored as zeros and blanks in ADABAS files, some SAS WHERE clauses are also impacted. For example, if either of the following SAS WHERE clauses are issued,

```
where aa is missing;
where aa is null;
```

the resulting condition is sent to ADABAS:

```
where aa = 0 (numeric)
where aa = ' ' (alphanumeric)
```

Note: Null values are processed differently by ADABAS if the ADABAS descriptor used in a WHERE clause has the Null Value Suppress (NU) definition option defined for it. Δ

Using Multiple-Value Fields in Selection Criteria

A multiple-value field can have 0 to 191 values per record, and ADABAS assigns an occurrence number to each value. When you include a multiple-value field in SAS/ACCESS descriptor files, you can use SAS variables that reference individual occurrences and a SAS variable that references all occurrences to perform special WHERE clause queries.

The following table lists whether you can use a multiple-value field or its corresponding SAS variables in the SAS and view WHERE clauses.

Table A2.2 Multiple-Value Fields in WHERE Clauses

Multiple-Value Field	SAS WHERE Clause	View WHERE Clause
ADABAS data field name	no	yes
SAS name for individual Occurrence variable	yes	no
_ANY variable	yes	yes

WHERE Clause Examples

Using the multiple-value data field BRANCH-OFFICE from the CUSTOMERS DDM, the following examples illustrate using a multiple-value field in WHERE clauses.

WHERE Clause Example 1

In a view WHERE clause, you can reference an ADABAS multiple-value field name, but you cannot do so in a SAS WHERE clause. For example, with the following WHERE clause in a view descriptor, the interface view engine searches all values of the multiple-value field:

```
where branch-office='LONDON'
```

The view WHERE clause produces the results in the following output.

Output A2.7 Results of ADABAS Multiple-Value Field Name in View WHERE Clause

OBS	CUSTNUM	BR_ANY	BRANCH_1	BRANCH_2	BRANCH_3	BRANCH_4
1	14324742		TORONTO	HOUSTON	TOKYO	LONDON
2	26422096		LONDON	NEW YORK		
3	26984578		LONDON	NEW YORK	ROME	
4	27654351		LONDON	BOSTON		
5	28710427		LONDON			

WHERE Clause Example 2

You can use the individual occurrence SAS variables created by the ACCESS procedure such as BRANCH_1, BRANCH_2, and so on, in SAS WHERE clauses, but you cannot use them in a view WHERE clause. Note that individual occurrence conditions must be processed by SAS after ADABAS has completed its selection processing.

For example, the following SAS WHERE clause searches the second occurrence for BRANCH-OFFICE and retrieves the London values. SAS post-processes all records returned from the interface view engine to see if they meet the SAS WHERE clause in effect.

```
where branch_1='LONDON'
```

The SAS WHERE clause produces the results in the following output.

Output A2.8 Results of Individual Occurrence SAS Variable in SAS WHERE Clause

OBS	CUSTNUM	BR_ANY	BRANCH_1	BRANCH_2	BRANCH_3	BRANCH_4
1	26422096		LONDON	NEW YORK		
2	26984578		LONDON	NEW YORK	ROME	
3	27654351		LONDON	BOSTON		
4	28710427		LONDON			

WHERE Clause Example 3

You can use the `_ANY` variable created by the `ACCESS` procedure in both a SAS `WHERE` clause and a view `WHERE` clause. However, if you use the `_ANY` variable in a SAS `WHERE` clause, the ADABAS interface view engine must be able to process the entire SAS `WHERE` clause.

For example, with the following `WHERE` clause, the engine searches all occurrences of the multiple-value field:

```
where br_any = 'LONDON'
```

Whether that `WHERE` clause is a SAS `WHERE` clause or a view `WHERE` clause, the results in the following output are produced. They are the same as for Output A2.7.

Output A2.9 Results of `_ANY` Variable in View or SAS `WHERE` Clause

OBS	CUSTNUM	BR_ANY	BRANCH_1	BRANCH_2	BRANCH_3	BRANCH_4
1	14324742		TORONTO	HOUSTON	TOKYO	LONDON
2	26422096		LONDON	NEW YORK		
3	26984578		LONDON	NEW YORK	ROME	
4	27654351		LONDON	BOSTON		
5	28710427		LONDON			

This functionality prevents you from having to enter repetitive selection criteria such as the following:

```
where branch_1='LONDON' or branch_2='LONDON'
      or branch_3='LONDON' ...
```

Periodic Group Fields in Selection Criteria

For an ADABAS periodic group data field, the `ACCESS` procedure automatically creates a SAS variable for the occurrence number within the periodic group. For example, the NATURAL DDM named `CUSTOMERS` has a periodic group field named `SIGNATURE-LIST`, which groups data fields `LIMIT` and `SIGNATURE`. The `ACCESS` procedure creates a SAS variable named `SL_OCCUR` for the occurrence numbers in `LIMIT` and `SIGNATURE`.

By including the `_OCCUR` variable in a view descriptor, you can retrieve the occurrence numbers for the periodic group. You can also include the `_OCCUR` variable in SAS `WHERE` clauses to qualify data, but the condition is processed by SAS after ADABAS has completed its selection processing. You cannot update the occurrence values, and you cannot use the `_OCCUR` variable in a view `WHERE` clause.

The following table lists whether you can use periodic group SAS variable names, periodic group occurrence syntax, and a periodic group's corresponding `_OCCUR` variable in SAS and view WHERE clauses.

Table A2.3 Periodic Group Fields in WHERE Clauses

Periodic Group Field	SAS WHERE	
	Clause	View WHERE Clause
SAS variable name	yes	yes
ADABAS data field name and occurrence syntax	no	yes
<code>_OCCUR</code> variable	yes	no

WHERE Clause Examples

Using the periodic group data field `LIMIT` from the `CUSTOMERS` DDM, the following examples illustrate using a periodic group data field in WHERE clauses.

WHERE Clause Example 1

You can use the SAS variable name of a data field within a periodic group in both a SAS WHERE clause and a view WHERE clause. However, they will not always produce the same results because the SAS WHERE clause post-processes the results and, using the following example, looks at the value of variable `LIMIT` to determine whether it's equal to 5000. The view WHERE clause is not post-processed; when you use a periodic group field, ADABAS qualifies all periodic group occurrence values if any one meets the WHERE clause criteria.

For example, you can include the following WHERE clause in a view descriptor, and you can issue it as a SAS WHERE clause:

```
where limit = 5000
```

Stored in a view descriptor, the WHERE clause produces the results in the following output:

Output A2.10 Results of Referencing a Periodic Group Data Field in View a WHERE Clause

OBS	CUSTNUM	SL_OCCUR	LIMIT
1	12345678	1	5000.00
2	14324742	1	5000.00
3	14324742	2	25000.00
4	14569877	1	5000.00
5	14569877	2	100000.00
6	19783482	1	5000.00
7	19783482	2	10000.00
8	26422096	1	5000.00
9	26422096	2	10000.00
10	27654351	1	5000.00
11	29834248	1	5000.00

However, as a SAS WHERE clause, the results in the following output are produced.

Output A2.11 Results of Referencing Periodic Group Data Field in SAS WHERE Clause

OBS	CUSTNUM	SL_OCCUR	LIMIT
1	12345678	1	5000.00
2	14324742	1	5000.00
3	14569877	1	5000.00
4	19783482	1	5000.00
5	26422096	1	5000.00
6	27654351	1	5000.00
7	29834248	1	5000.00
8	43459747	2	5000.00

WHERE Clause Example 2

You can qualify a specific occurrence of a periodic group with a view WHERE clause, but only by using the periodic group occurrence syntax. However, all of the periodic group occurrence values for the qualified records are returned, not just the individual occurrence specified in the view WHERE clause. You cannot specify the occurrence syntax in a SAS WHERE clause. For example, this view WHERE clause produces the results in the following output.

```
where limit(2) = 5000
```

Output A2.12 Results of Qualifying Periodic Group Occurrence Syntax in View WHERE Clause

OBS	CUSTNUM	SL_OCCUR	LIMIT
1	43459747	1	1000.00
2	43459747	2	5000.00

WHERE Clause Example 3

If you include the `_OCCUR` SAS variable in the view descriptor, you can use it in a SAS WHERE clause to specify an occurrence. However, you cannot use the `_OCCUR` variable in a view WHERE clause.

For example, this SAS WHERE clause produces the results shown in the following output.

```
where sl_occur = 2
```

Output A2.13 Results of Including _OCCUR Variable in SAS WHERE Clause

OBS	CUSTNUM	SL_OCCUR	LIMIT
1	14324742	2	25000.00
2	14569877	2	100000.00
3	14898029	2	50000.00
4	18543489	2	50000.00
5	19783482	2	10000.00
6	19876078	2	25000.00
7	26422096	2	10000.00
8	43459747	2	5000.00

To qualify the data even further, you could use this SAS WHERE clause, which produces the results in the following output.

```
where sl_occur = 2 and limit = 5000
```

Output A2.14 Results of Including _OCCUR Variable and Occurrence Syntax in SAS WHERE Clause

OBS	CUSTNUM	SL_OCCUR	LIMIT
1	43459747	2	5000.00

Using a SAS WHERE Clause for Selection Criteria

In addition to (or instead of) including a WHERE clause in your view descriptor for selection criteria, you can also specify a SAS WHERE clause in a SAS program for selection criteria.

When you specify a SAS WHERE clause, the SAS/ACCESS interface view engine translates those conditions into view WHERE clause conditions. Then, if the view descriptor includes a WHERE clause, the interface view engine connects the conditions with the Boolean operator AND. By default, the SAS WHERE clause conditions are connected before the view WHERE clause conditions. For example, if a view descriptor includes the condition

```
sex=female
```

and the SAS WHERE clause condition translates into

```
position=marketing
```

the resulting selection criteria are

```
(position=marketing) and (sex=female)
```

When the interface view engine translates SAS WHERE clause conditions into view WHERE clause conditions, some SAS WHERE clause capabilities are not available in a view WHERE clause. That is, some SAS WHERE clauses cannot be totally satisfied by the interface view engine.

To allow for this possibility, the interface view engine first evaluates the SAS WHERE clause and determines whether the conditions can be handled. The interface

view engine might be able to partially satisfy a SAS WHERE clause, as in the following example:

```
proc print data=vlib.empl;
  where lastname < 'KAP' and payrate > 30 * overtime;
run;
```

The interface view engine translates as much of the SAS WHERE clause as possible without producing incorrect results or a syntax error. In the example above, the engine has no problem with the first condition, but the arithmetic in the second condition is not supported. The interface view engine uses the condition *where lastname < 'KAP'* to filter out as many logical records as possible to improve performance.

Any conditions that are not supported are bypassed by the interface view engine, and post-processing (handled automatically by SAS) is required after the engine does its subsetting. The engine bypasses the following conditions:

- unacceptable conditions.
- conditions connected with OR to unacceptable conditions.

In the following table, assume DF1, DF2, and DF3 are ADABAS data fields referenced by a view descriptor. Remember that SAS never sees view WHERE clauses.

Table A2.4 Periodic Group Fields in WHERE Clauses

SAS WHERE Clause	View WHERE Clause	Translation	Processing Required?
DF2=B OR DF3>DF4+10	(DF1=A)	(DF1=A)	Yes
DF2=B & DF3>DF4+10	DF1=A	(DF2=B) & (DF1=A)	Yes
DF2=B & DF3>C	DF1=A	(DF2=B) & (DF3>C) & (DF1=A)	No
DF2=B OR DF3>C	DF1=A	(DF2=B) OR (DF3>C) & (DF1=A)	No

SAS WHERE Clause Conditions Acceptable to ADABAS

The following information explains how the interface view engine translates acceptable SAS WHERE clause conditions into view WHERE clause conditions.

- The operators are translated as shown in the following table.

Table A2.5 Acceptable SAS WHERE Clause Conditions in View WHERE Clause Conditions

SAS WHERE Clause Syntax	View WHERE Clause Translation
=	=
>	>
<	<
<>	!=
≥	≥
≤	≤

SAS WHERE Clause Syntax	View WHERE Clause Translation
((
))
AND	AND
OR	OR

- The interface view engine also translates BETWEEN and IN conditions and the date format (if a SAS format is supplied in the DB Content field).

Table A2.6 Translating BETWEEN and IN Conditions and the Date Format

SAS WHERE Clause Syntax	View WHERE Clause Translation
DF1 BETWEEN 1 AND 3	(DF1 >= 1 AND DF1 <= 3)
DF1 IN (4,9,14)	DF1=4 OR DF1=9 or DF1=14
DF4 = '02AUG87'D	DF4 = 870802

SAS WHERE Clause Conditions Not Acceptable to ADABAS

Any SAS WHERE clause conditions that are not acceptable to the ADABAS interface view engine are handled automatically by SAS post-processing. Following are some (but not all) of those conditions:

- item-to-item comparison
- pattern matching
- arithmetic expressions, for example,

```
WHERE DF1 = DF4 * 3
WHERE DF4 < - DF5
```

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

```
WHERE DF1
WHERE (DF1 = DF2) * 20
```

- concatenation of character variables
- truncated comparison, for example,

```
DF1 =: ABC
```

- DATETIME and TIME formats, for example,

```
'12:00'T
'01JAN60:12:00'DT
```

- SOUNDIX
- HAVING, GROUP BY
- NOT CONTAINS.

When a SAS WHERE Clause Must Reference Descriptor Data Fields

When you are using a SAS WHERE clause, a referenced ADABAS data field must be an ADABAS descriptor in the following situations:

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

Deciding How to Specify Selection Criteria

Use the following guidelines to determine when to use a SAS WHERE clause and when to use a view WHERE clause.

View WHERE Clause

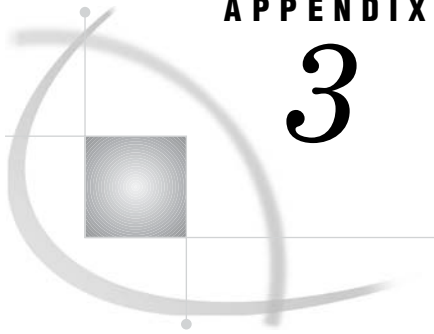
Include a WHERE clause in your view descriptor when you want to do the following:

- restrict users of view descriptors to certain subsets of data
- prevent users from sequentially passing all the ADABAS data
- use syntax not available in the SAS WHERE clause, such as periodic group occurrence syntax or multiple-value compares.

SAS WHERE Clause

Use a SAS WHERE clause when the previous guidelines do not apply and you want to meet the following criteria

- have more run-time flexibility in subsetting data
- use SAS WHERE clause capabilities that the view WHERE clause does not support, such as arithmetic expressions, truncated comparisons, or pattern matching
- use conditions on fabricated data fields such as ISN, periodic group _OCCUR variables, or any individually selected multiple-value fields
- combine AND and OR conditions using non-descriptor data fields.



APPENDIX

3

Example Data

<i>Introduction to the ADABAS Example Data</i>	132
<i>ADABAS Files</i>	132
<i>CUSTOMERS ADABAS File</i>	132
<i>EMPLOYEE ADABAS File</i>	138
<i>INVOICE ADABAS File</i>	140
<i>ORDER ADABAS File</i>	142
<i>NATURAL DDMs Based on the ADABAS Files</i>	145
<i>CUSTOMERS DDM</i>	145
<i>EMPLOYEE DDM</i>	145
<i>INVOICE DDM</i>	146
<i>ORDER DDM</i>	146
<i>Access Descriptors for ADABAS</i>	147
<i>Access Descriptors Based on ADABAS Files</i>	147
<i>ADLIB.CUSTOMER Access Descriptor</i>	147
<i>Access Descriptors Based on the NATURAL DDMs</i>	148
<i>MYLIB.CUSTS Access Descriptor</i>	148
<i>MYLIB.EMPLOYEE Access Descriptor</i>	148
<i>MYLIB.INVOICE Access Descriptor</i>	149
<i>MYLIB.ORDER</i>	149
<i>View Descriptors Based on the Access Descriptors for ADABAS</i>	150
<i>VLIB.ADAEMPS View Descriptor</i>	150
<i>VLIB.ALLEMP View Descriptor</i>	150
<i>VLIB.ALLORDR View Descriptor</i>	150
<i>VLIB.CUSORDR View Descriptor</i>	150
<i>VLIB.CUSPHON View Descriptor</i>	150
<i>VLIB.EMPINFO View Descriptor</i>	151
<i>VLIB.EMPS View Descriptor</i>	151
<i>VLIB.FORINV View Descriptor</i>	151
<i>VLIB.INV View Descriptor</i>	151
<i>VLIB.USACUST View Descriptor</i>	151
<i>VLIB.USAINV View Descriptor</i>	152
<i>VLIB.USAORDR View Descriptor</i>	152
<i>SAS Data Files for ADABAS</i>	152
<i>MYDATA.OUTOFSTK SAS Data File</i>	152
<i>MYDATA.SASEMPS SAS Data File</i>	153
<i>LIB6.BIRTHDAY Data File</i>	153

Introduction to the ADABAS Example Data

This appendix provides information about the ADABAS files, NATURAL DDMs, access descriptors, view descriptors, and SAS data files used in the examples in this document.

It shows the ADABAS data definition statements and the data that were used to build the ADABAS files. It also shows the access descriptors and view descriptors, along with any selection criteria. In addition, this appendix shows the data and the SAS statements that were used to create the SAS data files for the examples.

If you want to run the examples in this document, contact your SAS Software Representative for information about accessing the sample library files. The sample files contain instructions for creating the ADABAS files. The steps are as follows:

- 1 Create the ADABAS files using the ADABAS data definition statements.
- 2 Create the NATURAL DDMs for the ADABAS files as shown in each description.
- 3 Create the SAS data files.
- 4 Create an access descriptor and an associated view descriptor for each ADABAS file. Make sure that all SAS names match between the view descriptor and the ADABAS file. Use the access descriptors in this appendix as a model. Select every field for the access descriptors, and create views that also select every field.
- 5 Run the APPEND procedure with the data set options shown below. Use the SAS data file to update the view.

```
proc append data=SAS-file base=view-descriptor;
run;
```

ADABAS Files

This section describes the ADABAS files associated with the NATURAL DDMs that are used in this document's examples. For each file, the following information is provided:

- the ADABAS data definition statements used to create the ADABAS file
- the SAS DATA step used to create the SAS data file for populating the ADABAS file
- the example data.

The four ADABAS files used in the examples are named CUSTOMERS, EMPLOYEE, INVOICE, and ORDER.

CUSTOMERS ADABAS File

The CUSTOMERS file was created with the following ADABAS data definition statements:

```
//STEP01.DDCARD DD *
ADARUN PROGRAM=ADACMP
ADARUN DATABASE=001
ADARUN DEVICE=3380
ADARUN MODE=MULTI
ADARUN SVC=253
//STEP01.DDKARTE DD *
ADACMP COMPRESS
```

```

ADACMP FILE=45
ADACMP NUMREC=0
ADACMP FNDEF='01,CU,008,A,DE'
ADACMP FNDEF='01,SZ'
ADACMP FNDEF='02,ST,002,A,DE'
ADACMP FNDEF='02,ZI,005,U'
ADACMP FNDEF='01,CY,020,A,DE'
ADACMP FNDEF='01,PH,012,A'
ADACMP FNDEF='01,NA,060,A'
ADACMP FNDEF='01,CN,030,A'
ADACMP FNDEF='01,AD,040,A'
ADACMP FNDEF='01,CI,025,A'
ADACMP FNDEF='01,FO,006,U'
ADACMP FNDEF='01,SL,PE'
ADACMP FNDEF='02,LI,0014,U'
ADACMP FNDEF='02,SI,0030,A'
ADACMP FNDEF='01,BR,0025,A,MU(10)'
ADACMP SUPDE='SP=ST(1,2),ZI(1,2)'
ADACMP SUBDE='SB=ZI(1,2)'
//STEP02.DDCARD DD *
ADARUN PROGRAM=ADALOD
ADARUN DATABASE=001
ADARUN DEVICE=3380
ADARUN MODE=MULTI
ADARUN SVC=253
//STEP02.DDKARTE DD *
ADALOD LOAD FILE=45
ADALOD DSSIZE=5B
ADALOD NAME=CUSTOMERS
ADALOD MAXISN=100
ADALOD DSDEV=3380
ADALOD TEMPDEV=3380
ADALOD SORTSIZE=5
ADALOD TEMPSIZE=5

```

The DATA step is as follows:

```

data customer;
  /* customer number */
  input @1 custnum $8.
        @10 state $2.
        /* zipcode if company is */
        /* in the U.S.; otherwise */
        /* it is the mail code */
        /* appropriate for the */
        /* country where the */
        /* company is located */
        @13 zipcode 5.
        @20 country $20.
        @42 phone $12. /
  /* customer's company name*/
  @1 name $60. /
  /* contact at customer's */
  /* company */
  @1 contact $30.

```

```
@31 street      $40. /
@1  city        $25.
/* date of first order */
@30 firstord    yymmdd6./
/* signature limit #1 */
@1  limit       15.2
/* signature name #1 */
@20 signatur    $30. /
/* branch office #1 */
@1  branch_1    $25.
/* branch office #2 */
@30 branch_2    $25. /
/* branch office #3 */
@1  branch_3    $25.
/* branch office #4 */
@30 branch_4    $25.;
format firstord date7.;
datalines;
```

The data is shown in the following four outputs.

Output A3.1 Data in CUSTOMERS ADABAS File—Part 1

***** CUSTOMER DATA *****					
OBS	CUSTNUM	STATE	ZIPCODE	COUNTRY	PHONE
1	12345678	NC	.	USA	919/489-5682
2	14324742	CA	95123	USA	408/629-0589
3	14324742	CA	95123	USA	408/629-0589
4	14569877	NC	27514	USA	919/489-6792
5	14569877	NC	27514	USA	919/489-6792
6	14898029	MD	20850	USA	301/760-2541
7	14898029	MD	20850	USA	301/760-2541
8	14898029	MD	20850	USA	301/760-2541
OBS	NAME				
1					
2	SANTA CLARA VALLEY TECHNOLOGY SPECIALISTS				
3	SANTA CLARA VALLEY TECHNOLOGY SPECIALISTS				
4	PRECISION PRODUCTS				
5	PRECISION PRODUCTS				
6	UNIVERSITY BIOMEDICAL MATERIALS				
7	UNIVERSITY BIOMEDICAL MATERIALS				
8	UNIVERSITY BIOMEDICAL MATERIALS				
OBS	CONTACT			STREET	
1					
2	A. BAUM			5089 CALERO AVENUE	
3	A. BAUM			5089 CALERO AVENUE	
4	CHARLES BARON			198 FAYETTVILLE ROAD	
5	CHARLES BARON			198 FAYETTVILLE ROAD	
6	S. TURNER			1598 PICCARD DRIVE	
7	S. TURNER			1598 PICCARD DRIVE	
8	S. TURNER			1598 PICCARD DRIVE	
OBS	CITY	FIRSTORD	LIMIT	SIGNATUR	BRANCH_1
1		.	.		
2	SAN JOSE	05FEB65	5000	BOB HENSON	TORONTO
3	SAN JOSE	05FEB65	25000	KAREN DRESSER	TORONTO
4	MEMPHIS	15AUG83	5000	JEAN CRANDALL	NEW YORK
5	MEMPHIS	15AUG83	100000	STEVE BLUNTSEN	NEW YORK
6	ROCKVILLE	12NOV76	10000	MASON FOXWORTH	NEW YORK
7	ROCKVILLE	12NOV76	50000	DANIEL STEVENS	NEW YORK
8	ROCKVILLE	12NOV76	100000	ELIZABETH PATTON	NEW YORK
OBS	BRANCH_2	BRANCH_3	BRANCH_4		
1					
2	HOUSTON	TOKYO	LONDON		
3	HOUSTON	TOKYO	LONDON		
4	CHICAGO	LOS ANGELES			
5	CHICAGO	LOS ANGELES			
6	CHICAGO	DALLAS			
7	CHICAGO	DALLAS			
8	CHICAGO	DALLAS			

Output A3.2 Data in CUSTOMERS ADABAS File—Part 2

***** CUSTOMER DATA *****					
OBS	CUSTNUM	STATE	ZIPCODE	COUNTRY	PHONE
9	15432147	MI	49001	USA	616/582-3906
10	18543489	TX	78701	USA	512/478-0788
11	18543489	TX	78701	USA	512/478-0788
12	18543489	TX	78701	USA	512/478-0788
13	19783482	VA	22090	USA	703/714-2900
14	19783482	VA	22090	USA	703/714-2900
15	19876078	CA	93274	USA	209/686-3953
16	19876078	CA	93274	USA	209/686-3953
OBS	NAME				
9	GREAT LAKES LABORATORY EQUIPMENT MANUFACTURERS				
10	LONE STAR STATE RESEARCH SUPPLIERS				
11	LONE STAR STATE RESEARCH SUPPLIERS				
12	LONE STAR STATE RESEARCH SUPPLIERS				
13	TWENTY-FIRST CENTURY MATERIALS				
14	TWENTY-FIRST CENTURY MATERIALS				
15	SAN JOAQUIN SCIENTIFIC AND INDUSTRIAL SUPPLY, INC.				
16	SAN JOAQUIN SCIENTIFIC AND INDUSTRIAL SUPPLY, INC.				
OBS	CONTACT		STREET		
9	D.W. KADARAUCH		103 HARRIET STREET		
10	A. SILVERIA		5609 RIO GRANDE		
11	A. SILVERIA		5609 RIO GRANDE		
12	A. SILVERIA		5609 RIO GRANDE		
13	M.R. HEFFERNAN		4613 MICHAEL FARADAY DRIVE		
14	M.R. HEFFERNAN		4613 MICHAEL FARADAY DRIVE		
15	J.A. WHITTEN		1095 HIGHWAY 99 SOUTH		
16	J.A. WHITTEN		1095 HIGHWAY 99 SOUTH		
OBS	CITY	FIRSTORD	LIMIT	SIGNATUR	BRANCH_1
9	KALAMAZOO	28APR86	10000	JACK TREVANE	CHICAGO
10	AUSTIN	10SEP79	10000	NANCY WALSH	HOUSTON
11	AUSTIN	10SEP79	50000	TED WHISTLER	HOUSTON
12	AUSTIN	10SEP79	100000	EVAN MASSEY	HOUSTON
13	RESTON	18JUL68	5000	PETER THOMAS	WASHINGTON D.C.
14	RESTON	18JUL68	10000	LOUIS PICKERING	WASHINGTON D.C.
15	TULARE	11MAY79	7500	EDWARD LOWE	
16	TULARE	11MAY79	25000	E.F. JENSEN	
OBS	BRANCH_2	BRANCH_3	BRANCH_4		
9	COLUMBUS				
10	DALLAS	EL PASO	LUBBOCK		
11	DALLAS	EL PASO	LUBBOCK		
12	DALLAS	EL PASO	LUBBOCK		
13	NEW YORK				
14	NEW YORK				
15					
16					

Output A3.3 Data in CUSTOMERS ADABAS File—Part 3

***** CUSTOMER DATA *****					
OBS	CUSTNUM	STATE	ZIPCODE	COUNTRY	PHONE
17	24589689		.	Yugoslavia	(012)736-202
18	26422096		75014	France	4268-54-72
19	26422096		75014	France	4268-54-72
20	26984578		5110	Austria	43-57-04
21	27654351		5010	Belgium	02/215-37-32
22	28710427	HV	3607	Netherlands	(021)570517
23	29834248		.	Britain	(0552)715311
24	31548901	BC	.	Canada	406/422-3413
OBS	NAME				
17	CENTAR ZA TECHNICKU I NAUCNU RESTAURIRANJE UMJETNINA				
18	SOCIETE DE RECHERCHES POUR DE CHIRURGIE ORTHOPEDIQUE				
19	SOCIETE DE RECHERCHES POUR DE CHIRURGIE ORTHOPEDIQUE				
20	INSTITUT FUR TEXTIL-FORSCHUNGS				
21	INSTITUT DE RECHERCHE SCIENTIFIQUE MEDICALE				
22	ANTONIE VAN LEEUWENHOEK VERENIGING VOOR MICROBIOLOGIE				
23	BRITISH MEDICAL RESEARCH AND SURGICAL SUPPLY				
24	NATIONAL COUNCIL FOR MATERIALS RESEARCH				
OBS	CONTACT			STREET	
17	J.V. VUKASINOVIC			TAKOVSKA 4	
18	Y. CHAVANON			40 RUE PERIGNON	
19	Y. CHAVANON			40 RUE PERIGNON	
20	GUNTER SPIELMANN			MECHITARISTENGASSE 5	
21	I. CLEMENS			103 RUE D'EGMONT	
22	M.C. BORGSTEEDE			BIRMOERSTRAAT 34	
23	A.D.M. BRYCESON			44 PRINCESS GATE, HYDE PARK	
24	W.E. MACDONALD			5063 RICHMOND MALL	
OBS	CITY	FIRSTORD	LIMIT	SIGNATUR	BRANCH_1
17	BELGRADE	30NOV81	.		
18	LA ROCHELLE	14JUN83	5000	MICHELE PICARD	LONDON
19	LA ROCHELLE	14JUN83	10000	M.L. SEIS	LONDON
20	VIENNA	25MAY87	100000	FRANZ BECH	LONDON
21	BRUSSELS	14OCT86	5000	C.J. HELMER	LONDON
22	THE HAGUE	10OCT85	10000	J.J. JASPER	LONDON
23	LONDON, SW7 1PU	29JAN86	5000	ELVIN POMEROY	SINGAPORE
24	VANCOUVER, V5T 1L2	19MAR84	1000	DAPHNE MARSHALL	SEATTLE
OBS	BRANCH_2	BRANCH_3		BRANCH_4	
17					
18	NEW YORK				
19	NEW YORK				
20	NEW YORK	ROME			
21	BOSTON				
22					
23	TORONTO	CAIRO			
24	TORONTO				

Output A3.4 Data in CUSTOMERS ADABAS File—Part 4

***** CUSTOMER DATA *****					
OBS	CUSTNUM	STATE	ZIPCODE	COUNTRY	PHONE
25	38763919		1405	Argentina	244-6324
26	39045213	SP	1051	Brazil	012/302-1021
27	43290587		.	Japan	(02)933-3212
28	43459747		3181	Australia	03/734-5111
29	43459747		3181	Australia	03/734-5111
30	46543295		.	Japan	(03)022-2332
31	46783280		2374	Singapore	3762855
32	48345514		.	United Arab Emirates	213445
OBS	NAME				
25	INSTITUTO DE BIOLOGIA Y MEDICINA NUCLEAR				
26	LABORATORIO DE PESQUISAS VETERINARIAS DESIDERIO FINAMOR				
27	HASSEI SAIBO GAKKAI				
28	RESEARCH OUTFITTERS				
29	RESEARCH OUTFITTERS				
30	WESTERN TECHNOLOGICAL SUPPLY				
31	NGEE TECHNOLOGICAL INSTITUTE				
32	GULF SCIENTIFIC SUPPLIES				
OBS	CONTACT		STREET		
25	JORGE RUNNAZZO		SALGUERO 2345		
26	ELISABETE REGIS GUILLAUMON		RUA DONA ANTONIA DE QUEIROS 381		
27	Y. FUKUDA		3-2-7 ETCHUJMA, KOTO-KU		
28	R.G. HUGHES		191 LOWER PLENTY ROAD		
29	R.G. HUGHES		191 LOWER PLENTY ROAD		
30			4-3-8 ETCHUJMA, KOTO-KU		
31	LING TAO SOON		356 CLEMENTI ROAD		
32	J.Q. RIFAI		POB 8032		
OBS	CITY	FIRSTORD	LIMIT	SIGNATUR	BRANCH_1
25	BUENOS AIRES	10DEC84	2500	M.L. CARLOS	MIAMI
26	SAO PAULO	18AUG82	1500	RICK ESTABAN	MIAMI
27	TOKYO 101	08FEB74	10000	R. YAMOTO	SAN FRANCISCO
28	PRAHRAN, VICTORIA	28JUL72	1000	DENNIS RICHMOND	SEATTLE
29	PRAHRAN, VICTORIA	28JUL72	5000	JANICE HEATH	SEATTLE
30	TOKYO 102	19APR84	10000	DAPHNE MARSHALL	SEATTLE
31	SINGAPORE	27SEP79	.		
32	RAS AL KHAIMAH	10SEP86	.		
OBS	BRANCH_2	BRANCH_3	BRANCH_4		
25	NEW YORK				
26	NEW YORK				
27					
28					
29					
30	TORONTO	SAN FRANCISCO	DENVER		
31					
32					

EMPLOYEE ADABAS File

The EMPLOYEE ADABAS file was created with the following ADABAS data definition statements:

```
//STEP01.DDCARD DD *
ADARUN PROGRAM=ADACMP
```

```

ADARUN DATABASE=001
ADARUN DEVICE=3380
ADARUN MODE=MULTI
ADARUN SVC=253
//STEP01.DDKARTE DD *
ADACMP COMPRESS
ADACMP FILE=46
ADACMP NUMREC=0
ADACMP FNDEF='01,ID,006,U,DE'
ADACMP FNDEF='01,HD,006,U'
ADACMP FNDEF='01,SA,007,U'
ADACMP FNDEF='01,DP,006,A'
ADACMP FNDEF='01,JC,005,U,DE'
ADACMP FNDEF='01,SX,001,A'
ADACMP FNDEF='01,BD,006,U'
ADACMP FNDEF='01,LN,018,A,DE'
ADACMP FNDEF='01,FN,015,A'
ADACMP FNDEF='01,MN,015,A'
ADACMP FNDEF='01,PH,004,A'
//STEP02.DDCARD DD *
ADARUN PROGRAM=ADALOD
ADARUN DATABASE=001
ADARUN DEVICE=3380
ADARUN MODE=MULTI
ADARUN SVC=253
//STEP02.DDKARTE DD *
ADALOD LOAD FILE=46
ADALOD DSSIZE=5B
ADALOD NAME=EMPLOYEE
ADALOD MAXISN=100
ADALOD DSDEV=3380
ADALOD TEMPDEV=3380
ADALOD SORTSIZE=5
ADALOD TEMPSIZE=5

```

The DATA step is as follows:

```

data employ;
  /* employee id number */
  input @1 empid 6.
        @10 hiredate yymmdd6.
        @20 salary 8.2
        @30 dept $6.
        @40 jobcode 5.
        @47 sex $1.
        @50 birthdat yymmdd6. /
        @1 lastname $18.
        @20 firstnam $15.
        @40 middlena $15.
        @60 phone $4. ;
  format hiredate date7.;
  format birthdat date7.;
  datalines;

```

The data is shown in the following output.

Output A3.5 Data for EMPLOYEE ADABAS File

EMPID	HIREDATE	SALARY	DEPT	JOBCODE	SEX	BIRTHDAT
119012	01JUL68	42340.58	CSR010	602	F	05JAN46
120591	05DEC80	31000.55	SHP002	602	F	12FEB46
123456	04APR89
127845	16JAN67	75320.34	ACC024	204	M	25DEC43
129540	01AUG82	56123.34	SHP002	204	F	31JUL60
135673	15JUL84	46322.58	ACC013	602	F	21MAR61
212916	15FEB51	52345.58	CSR010	602	F	29MAY28
216382	15JUN85	34004.65	SHP013	602	F	24JUL63
234967	19DEC88	17000.00	CSR004	602	M	21DEC67
237642	01NOV76	43200.34	SHP013	602	M	13MAR54
239185	07MAY81	57920.66	ACC024	602	M	28AUG59
254896	04APR85	35000.74	CSR011	204	M	06APR49
321783	10SEP67	48931.58	CSR011	602	M	03JUN35
328140	10JAN75	75000.34	ACC043	1204	F	02JUN51
346917	02MAR87	46000.33	SHP013	204	F	15MAR50
356134	14JUN85	62450.75	ACC013	204	F	25OCT60
423286	19DEC88	32870.66	ACC024	602	M	31OCT64
456910	14JUN78	45000.58	CSR010	602	M	24SEP53
456921	19AUG87	33210.04	SHP002	602	M	12MAY62
457232	15JUL85	55000.66	ACC013	602	M	15OCT63
459287	02NOV64	50000.00	SHP024	204	M	05JAN34
677890	12DEC88	37610.00	CSR010	204	F	24APR65
LASTNAME	FIRSTNAM	MIDDLENA	PHONE			
WOLF-PROVENZA	G.	ANDREA	3467			
HAMMERSTEIN	S.	RACHAEL	3287			
VARGAS	PAUL	JESUS				
MEDER	VLADIMIR	JORAN	6231			
CHOU LAI	CLARA	JANE	3921			
HEMESLY	STEPHANIE	J.	6329			
WACHBERGER	MARIE-LOUISE	TERESA	8562			
PURINTON	PRUDENCE	VALENTINE	3852			
SMITH	GILBERT	IRVINE	7274			
BATTERSBY	R.	STEPHEN	8342			
DOS REMEDIOS	LEONARD	WESLEY	4892			
TAYLOR-HUNYADI	ITO	MISHIMA	0231			
GONZALES	GUILLEMO	RICARDO	3642			
MEDINA-SIDONIA	MARGARET	ROSE	5901			
SHIEKELESLAM	SHALA	Y.	8745			
DUNNETT	CHRISTINE	MARIE	4213			
MIFUNE	YUKIO	TOSHIRO	3278			
ARDIS	RICHARD	BINGHAM	4351			
KRAUSE	KARL-HEINZ	G.	7452			
LOVELL	WILLIAM	SINCLAIR	6321			
RODRIGUES	JUAN	M.	5879			
NISHIMATSU-LYNCH	CAROL	ANNE	6245			

INVOICE ADABAS File

The INVOICE ADABAS file was created with the following ADABAS data definition statements:

```
//STEP01.DDCARD DD *
ADARUN PROGRAM=ADACMP
ADARUN DATABASE=001
ADARUN DEVICE=3380
ADARUN MODE=MULTI
ADARUN SVC=253
//STEP01.DDKARTE DD *
```

```

ADACMP COMPRESS
ADACMP FILE=47
ADACMP NUMREC=0
ADACMP FNDEF='01,IV,005,U,DE'
ADACMP FNDEF='01,BT,008,A'
ADACMP FNDEF='01,AM,014,U,DE'
ADACMP FNDEF='01,CY,020,A,DE'
ADACMP FNDEF='01,AU,010,U'
ADACMP FNDEF='01,BB,006,U,DE'
ADACMP FNDEF='01,BO,006,U'
ADACMP FNDEF='01,PO,006,U,DE'
ADACMP FNDEF='01,CX,008,G'
//STEP02.DDCARD DD *
ADARUN PROGRAM=ADALOD
ADARUN DATABASE=001
ADARUN DEVICE=3380
ADARUN MODE=MULTI
ADARUN SVC=253
//STEP02.DDKARTE DD *
ADALOD LOAD FILE=47
ADALOD DSSIZE=5B
ADALOD NAME=INVOICE
ADALOD MAXISN=100
ADALOD DSDEV=3380
ADALOD TEMPDEV=3380
ADALOD SORTSIZE=5
ADALOD TEMPSIZE=5

```

The DATA step is as follows:

```

data invoice;
  /* invoice number */
  input @1 invoicen 5.
        /* company that placed the order */
        @7 billedto $8.

        /* amount of bill in local currency */
        @15 amtbille 15.2
        @30 country $20.
        /* amount of bill in U.S. dollars */
        @50 amountin 11.2 /

        /* employee who wrote the bill */
        @1 billedby 6.

        /* date that bill was sent */
        @10 billedon yymmdd6.

        /* date that bill was paid */
        @20 paidon yymmdd6.

        /* time of day that the */
        /* exchange rate to U.S. */
        /* dollars was computed */

```

```

@30 computed time8. ;
format billedon date7.;
format paidon date7.;
datalines;

```

The data is shown in the following output.

Output A3.6 Data for ORDER ADABAS File

INVOICEN	BILLEDTO	AMTBILLE	COUNTRY	AMOUNTIN
11270	39045213	1340738760.9	Brazil	2256870.0
11271	18543489	11063836.0	USA	11063836.0
11273	19783482	252148.5	USA	252148.5
11276	14324742	1934460.0	USA	1934460.0
11278	14898029	1400825.0	USA	1400825.0
11280	39045213	1340738760.9	Brazil	2256870.0
11282	19783482	252148.5	USA	252148.5
11285	38763919	34891210.2	Argentina	2256870.0
11286	43459747	12679156.0	Australia	11063836.0
11287	15432147	252148.5	USA	252148.5
12051	39045213	1340738760.9	Brazil	2256870.0
12102	18543489	11063836.0	USA	11063836.0
12263	19783482	252148.5	USA	252148.5
12468	14898029	1400825.0	USA	1400825.0
12471	39045213	1340738760.9	Brazil	2256870.0
12476	38763919	34891210.2	Argentina	2256870.0
12478	15432147	252148.5	USA	252148.5
BILLEDON	PAIDON	COMPUTED		
239185	05OCT88	18OCT88	39646	
457232	05OCT88	11OCT88	.	
239185	06OCT88	11NOV88	.	
135673	06OCT88	20OCT88	.	
239185	06OCT88	19OCT88	.	
423286	07OCT88	20OCT88	58154	
457232	07OCT88	25OCT88	.	
239185	10OCT88	30NOV88	55163	
423286	10OCT88	.	33827	
457232	11OCT88	04NOV88	.	
457232	02NOV88	.	31185	
239185	17NOV88	.	.	
423286	05DEC88	.	.	
135673	24DEC88	02JAN89	.	
457232	27DEC88	.	50945	
135673	24DEC88	.	39563	
423286	24DEC88	02JAN89	.	

ORDER ADABAS File

The ORDER ADABAS file was created with the following ADABAS data definition statements:

```

//STEP01.DDCARD DD *
ADARUN PROGRAM=ADACMP
ADARUN DATABASE=001
ADARUN DEVICE=3380
ADARUN MODE=MULTI
ADARUN SVC=253
//STEP01.DDKARTE DD *
ADACMP COMPRESS

```

```

ADACMP FILE=48
ADACMP NUMREC=0
ADACMP FNDEF='01,ON,005,U,DE'
ADACMP FNDEF='01,SN,004,U'
ADACMP FNDEF='01,LN,004,U'
ADACMP FNDEF='01,FC,010,U'
ADACMP FNDEF='01,ST,008,A,DE'
ADACMP FNDEF='01,DO,006,U'
ADACMP FNDEF='01,DS,006,U'
ADACMP FNDEF='01,TB,006,U'
ADACMP FNDEF='01,PB,006,U'
ADACMP FNDEF='01,SF,001,A'
//STEP02.DDCARD DD *
ADARUN PROGRAM=ADALOD
ADARUN DATABASE=001
ADARUN DEVICE=3380
ADARUN MODE=MULTI
ADARUN SVC=253
//STEP02.DDKARTE DD *
ADALOD LOAD FILE=48
ADALOD DSSIZE=5B
ADALOD NAME=ORDER
ADALOD MAXISN=100
ADALOD DSDEV=3380
ADALOD TEMPDEV=3380
ADALOD SORTSIZE=5
ADALOD TEMPSIZE=5

```

The DATA step is as follows:

```

data orders;
  /* order number */
  input @1 ordernum 5.
        /* stock number */
        @6 stocknum 4.

        /* length of material ordered */
        @10 length 4.

        /* fabric charges */
        @15 fabricch 11.2

        /* customer whom order is to be
        /* shipped to */
        @27 shipto $8.

        /* date of order */
        @35 dateorde yymmdd6.
        /* date that order was shipped */
        @45 shipped yymmdd6.

        /* employee who took the order */
        @55 takenby 6.

```

```

/* employee who processed the order */
@62 processe 6.

/* this flag signals that      */
/* special instructions are     */
/* associated with this order. */
@69 speciali $1. ;
format dateorde date7.;
format shipped date7.;
datalines;

```

The data is shown in the following output.

Output A3.7 Data for ORDER ADABAS File

O	S	F	D	P	S				
R	T	A	A	S	T				
D	O	L	B	S	T				
E	C	E	R	H	A				
R	K	N	I	I	O				
N	N	G	C	P	R				
U	U	T	C	T	D				
M	M	H	H	O	E				
11269	9870	690	.	19876078	03OCT88	.	212916	.	
11270	1279	1750	2256870.0	39045213	03OCT88	19OCT88	321783	237642	X
11271	8934	110	11063836.0	18543489	03OCT88	13OCT88	456910	456921	
11272	3478	1000	.	29834248	03OCT88	.	234967	.	
11273	2567	450	252148.5	19783482	04OCT88	14NOV88	119012	216382	
11274	4789	1000	.	15432147	04OCT88	.	212916	.	
11275	3478	1000	.	29834248	04OCT88	.	234967	.	
11276	1279	1500	1934460.0	14324742	04OCT88	21OCT88	321783	120591	X
11277	8934	100	10058033.0	31548901	05OCT88	.	456910	.	
11278	2567	2500	1400825.0	14898029	05OCT88	20OCT88	119012	456921	
11279	9870	650	.	48345514	05OCT88	.	212916	.	
11280	1279	1750	2256870.0	39045213	06OCT88	21OCT88	321783	237642	X
11281	8934	110	11063836.0	18543489	06OCT88	27OCT88	456910	216382	
11282	2567	450	252148.5	19783482	06OCT88	26OCT88	119012	456921	
11283	9870	690	.	18543489	07OCT88	.	212916	.	
11284	3478	1000	.	24589689	07OCT88	.	234967	.	
11285	1279	1750	2256870.0	38763919	07OCT88	02DEC88	321783	120591	X
11286	8934	110	11063836.0	43459747	07OCT88	03NOV88	456910	237642	
11287	2567	450	252148.5	15432147	07OCT88	07NOV88	119012	216382	
11288	9870	690	.	14324742	10OCT88	.	212916	.	Y
11969	9870	690	.	19876078	25OCT88	.	212916	.	
12051	1279	1750	2256870.0	39045213	31OCT88	.	321783	.	X
12102	8934	110	11063836.0	18543489	15NOV88	.	456910	.	
12160	3478	1000	.	29834248	19NOV88	.	234967	.	Z
12263	2567	450	252148.5	19783482	01DEC88	.	119012	.	
12464	4789	1000	.	15432147	23DEC88	.	212916	.	
12465	3478	1000	.	29834248	23DEC88	.	234967	.	
12466	1279	1500	1934460.0	14324742	23DEC88	.	321783	.	X
12467	8934	100	10058033.0	31548901	23DEC88	.	456910	.	
12468	2567	2500	1400825.0	14898029	23DEC88	03JAN89	119012	120591	
12470	9870	650	.	48345514	23DEC88	.	212916	.	
12471	1279	1750	2256870.0	39045213	23DEC88	.	321783	.	X
12472	8934	110	11063836.0	18543489	23DEC88	03JAN89	456910	237642	
12473	2567	450	252148.5	19783482	23DEC88	.	119012	.	
12474	9870	690	.	18543489	23DEC88	.	212916	.	
12475	3478	1000	.	24589689	23DEC88	.	234967	.	
12476	1279	1750	2256870.0	38763919	23DEC88	03JAN89	321783	456921	X
12477	8934	110	11063836.0	43459747	23DEC88	.	456910	.	
12478	2567	450	252148.5	15432147	23DEC88	03JAN89	119012	216382	
12479	9870	690	.	14324742	23DEC88	.	212916	.	

NATURAL DDMs Based on the ADABAS Files

This section shows descriptions of the NATURAL DDMs created for the preceding ADABAS files. The DDMS are presented in alphabetical order.

CUSTOMERS DDM

The CUSTOMERS DDM contains the description in the following output.

Output A3.8 CUSTOMERS DDM

VIEW	DB NAME	DEF.SEQ:	DBID:1	FNR:	45
COMMAND:					
I T L	DB NAME	F LENG	S D	REMARK	
-----bottom-----					
1	CU CUSTOMER	A 8.0	D		
G 1	SZ STATEZIP				
2	ST STATE	A 2.0	D		
2	ZI ZIPCODE	N 5.0			
1	CY COUNTRY	A 20.0	D		
1	PH TELEPHONE	A 12.0			
1	NA NAME	A 60.0			
1	CN CONTACT	A 30.0			
1	AD STREETADDRESS	A 40.0			
1	CI CITY	A 25.0			
1	FO FIRSTORDERDATE	N 6.0			
P 1	SL SIGNATURE-LIST				
2	LI LIMIT	N 14.2			
2	SI SIGNATURE	A 30.0			
M 1	BR BRANCH-OFFICE	A 25.0			
1	SP STATE-ZIPLAST2	A 4.0			
1	SB ZIPLAST2	N 2.0			

EMPLOYEE DDM

The EMPLOYEE DDM contains the description shown in the following output.

Output A3.9 EMPLOYEE DDM

```

VIEW      : EMPLOYEE                      DEF.SEQ:  DBID:1  FNR:  46
COMMAND:
I T L DB NAME                          F LENG  S D      REMARK
-----all-----
  1 ID EMPID                            N 6.0    D
  1 HD HIREDATE                          N 6.0
  1 SA SALARY                             N 7.2
  1 DP DEPT                               A 6.0
  1 JC JOBCODE                            N 5.0    D
  1 SX SEX                                A 1.0
  1 BD BIRTHDATE                          N 6.0
  1 LN LASTNAME                           A 18.0   D
  1 FN FIRSTNAME                          A 15.0
  1 MN MIDDLENAME                          A 15.0
  1 PH PHONE                              A 4.0

```

INVOICE DDM

The INVOICE DDM contains the description shown in the following output.

Output A3.10 INVOICE DDM

```

VIEW      : INVOICE                      DEF.SEQ:  DBID:1  FNR:  47
COMMAND:
I T L DB NAME                          F LENG  S D      REMARK
-----all-----
  1 IB INVOICENUM                        N 5.0    D
  1 BT BILLEDTO                          A 8.0
  1 AM AMTBILLED                          N 14.2   D
  1 CY COUNTRY                            A 20.0   D
  1 AU AMOUNTINUS                         N 10.2
  1 BB BILLEDBY                           N 6.0    D
  1 BO BILLEDON                           N 6.0
  1 PO PAIDON                             N 6.0    D
  1 CX COMPUTEREXCHANGE                  F 8.0    F

```

ORDER DDM

The ORDER DDM contains the description shown in the following output.

Output A3.11 ORDER DDM

VIEW	DB NAME	F	LENG	S	D	REMARK
VIEW : ORDER						
COMMAND:						
I T L DB NAME						
-----all-----						
1 ON ORDERNUM		N	5.0		D	
1 SN STOCKNUM		N	4.0			
1 LN LENGTH		N	4.0			
1 FC FABRICCHARGES		N	10.2			
1 ST SHIPTO		A	8.0		D	
1 DO DATEORDERED		N	6.0			
1 DS SHIPPED		N	6.0			
1 TB TAKENBY		N	6.0			
1 PB PROCESSEDBY		N	6.0			
1 SF SPECIALINSTRUCTION		A	1.0			

Access Descriptors for ADABAS

An ADABAS access descriptor can be based on an ADABAS file or on a NATURAL DDM.

Access Descriptors Based on ADABAS Files

This section shows an access descriptor definition that is based on an ADABAS file and the same access descriptor definition based on the CUSTOMER NATURAL DDM.

ADLIB.CUSTOMER Access Descriptor

The ADLIB.CUSTOMER access descriptor was created as follows:

```
proc access dbms=adabas;
  create adlib.customer.access;
  adbfile(number=15 password=cuspw
    cipher=cuscc dbid=1);
  sysfile(number=15 password=cuspwsys
    cipher=cusccsys dbid=1);
  secfile(number=16 password=cuspwsec
    cipher=cusccsec dbid=1);
  assign=yes;
  rename cu = custnum
    ph = phone
    ad = street;
  format fo = date7.;
  informat fo = date7.;
  content fo = yymmdd6.;
  mvf br occurs = 4
run;
```

By specifying an ADABAS file number instead of a DDM, the definition bypasses NATURAL SECURITY. The following is an example of the same access descriptor written to use NATURAL SECURITY:

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

Access Descriptors Based on the NATURAL DDMs

This section shows the access descriptors used in this document that are based on NATURAL DDMs. All of the view descriptors in this document were created from these access descriptors. The access descriptors are presented in alphabetical order.

MYLIB.CUSTS Access Descriptor

The MYLIB.CUSTS access descriptor was created as follows:

```
proc access dbms=adabas;
  create mylib.custs.access;
  nss(library=demo user=demol password=demol);
  sysfile(number=15 dbid=1);
  secfile(number=16 dbid=1);
  ddm=customers;
  assign=yes;
  drop contact;
  rename customer = custnum
    telephone = phone
    streetaddress = street;
  format firstorderdate = date7.;
  informat firstorderdate = date7.;
  content firstorderdate = yymmdd6.;
  mvf "BRANCH-OFFICE" occurs = 4;
run;
```

MYLIB.EMPLOYEE Access Descriptor

The MYLIB.EMPLOYEE access descriptor was created as follows:

```
proc access dbms=adabas;
  create mylib.employee.access;
```

```

nss(library=demo user=demol password=demol);
sysfile(number=15 dbid=1);
secfile(number=16 dbid=1);
ddm=employee;
assign=yes;
format hiredate = date7.;
informat hiredate = date7.;
content hiredate = yymmdd6.;
format birthdate = date7.;
informat birthdate = date7.;
content birthdate = yymmdd6.;
run;

```

MYLIB.INVOICE Access Descriptor

The MYLIB.INVOICE access descriptor was created as follows:

```

proc access dbms=adabas;
  create mylib.invoice.access;
  nss(library=demo user=demol password=demol);
  sysfile(number=15 dbid=1);
  secfile(number=16 dbid=1);
  ddm=invoice;
  assign=yes;
  format billedon = date7.;
  informat billedon = date7.;
  content billedon = yymmdd6.;
  format paidon = date7.;
  informat paidon = date7.;
  content paidon = yymmdd6.;
run;

```

MYLIB.ORDER

The MYLIB.ORDER access descriptor was created as follows:

```

proc access dbms=adabas;
  create mylib.order.access;
  nss(library=demo user=demol password=demol);
  sysfile(number=15 dbid=1);
  secfile(number=16 dbid=1);
  ddm=order;
  assign=yes;
  format dateordered = date7.;
  informat dateordered = date7.;
  content dateordered = yymmdd6.;
  format shipped = date7.;
  informat shipped = date7.;
  content shipped = yymmdd6.;
run;

```

View Descriptors Based on the Access Descriptors for ADABAS

This section shows the view descriptors used in this document to access ADABAS data. The view descriptors are presented in alphabetical order.

VLIB.ADAEMPS View Descriptor

The VLIB.ADAEMPS view descriptor was created as follows:

```
proc access dbms=adabas ad=mylib.employee;
  create vlib.adaemps.view;
  select empid birthdate;
  select lastname firstname middlename;
run;
```

VLIB.ALLEMP View Descriptor

The VLIB.ALLEMP view descriptor was created as follows:

```
proc access dbms=adabas ad=mylib.employee;
  create vlib.allemp.view;
  select all;
  reset isn;
run;
```

VLIB.ALLORDR View Descriptor

The VLIB.ALLORDR view descriptor was created as follows:

```
proc access dbms=adabas ad=mylib.order;
  create vlib.allordr.view;
  select all;
  reset isn;
run;
```

VLIB.CUSORDR View Descriptor

The VLIB.CUSORDR view descriptor was created as follows:

```
proc access dbms=adabas ad=mylib.order;
  create vlib.cusordr.view;
  select stocknum shipto;
run;
```

VLIB.CUSPHON View Descriptor

The VLIB.CUSPHON view descriptor was created as follows:

```
proc access dbms=adabas ad=mylib.custs;
  create vlib.cusphon.view;
  select customer telephone name;
run;
```

VLIB.EMPINFO View Descriptor

The VLIB.EMPINFO view descriptor was created as follows:

```
proc access dbms=adabas ad=mylib.employee;
  create vlib.empinfo.view;
  select empid dept lastname;
run;
```

VLIB.EMPS View Descriptor

The VLIB.EMPS view descriptor was created as follows. This descriptor includes sort and WHERE statements to specify selection criteria.

```
proc access dbms=adabas ad=mylib.employee;
  create vlib.emps.view;
  select empid jobcode birthdate lastname;
  subset where jobcode = 602;
  subset sort lastname;
run;
```

VLIB.FORINV View Descriptor

The VLIB.FORINV view descriptor was created as follows. This descriptor includes a WHERE statement to specify selection criteria.

```
proc access dbms=adabas ad=mylib.invoice;
  create vlib.forinv.view;
  select all;
  reset isn computedexchange;
  subset where country != 'USA';
run;
```

VLIB.INV View Descriptor

The VLIB.INV view descriptor was created as follows. This descriptor includes a sort statement to specify selection criteria.

```
proc access dbms=adabas ad=mylib.invoice;
  create vlib.inv.view;
  select invoicenum amtbilled country
    billedby paidon;
  subset sort billedby;
run;
```

VLIB.USACUST View Descriptor

The VLIB.USACUST view descriptor was created as follows. This descriptor includes SORT and WHERE statements to specify selection criteria.

```
proc access dbms=adabas ad=mylib.custs;
  create vlib.usacust.view;
  select all;
```

```

reset isn telephone streetaddress
    city "STATE-ZIPLAST2" ziplast2;
mvf "BRANCH-OFFICE" reset br_any
    branch_1 branch_3 branch_4;
subset where country = 'USA';
subset sort customer;
run;

```

VLIB.USAINV View Descriptor

The VLIB.USAINV view descriptor was created as follows. This descriptor includes a WHERE statement to specify selection criteria.

```

proc access dbms=adabas ad=mylib.invoice;
    create vlib.usainv.view;
    select all;
    reset isn computedexchange;
    subset where country = 'USA';
run;

```

VLIB.USAORDR View Descriptor

The VLIB.USAORDR view descriptor was created as follows. This view descriptor uses a SORT statement to specify selection criteria.

```

proc access dbms=adabas ad=mylib.order;
    create vlib.usaodr.view;
    select ordernum stocknum length
        fabriccharges shipto;
    subset sort shipto;
run;

```

SAS Data Files for ADABAS

This section describes the SAS data files used in this document. It provides the SAS statements that created each data file and shows the output with the PRINT procedure.

MYDATA.OUTOFSTK SAS Data File

The SAS data file MYDATA.OUTOFSTK is used in Chapter 3, "ADABAS Data in SAS Programs," on page 17. It was created with the following SAS statements:

```

libname mydata 'your-SAS-library';
data mydata.outofstk;
    input fibernam $8. /* fiber name */
        fibernum; /* fiber number */
    datalines;
olefin 3478
gold 8934
dacron 4789
;
run;

```

The following PRINT procedure lists the data shown in Output 3.11.

```
proc print data=mydata.outofstk;
  title 'SAS Data File MYDATA.OUTOFSTK';
run;
```

MYDATA.SASEMPS SAS Data File

The SAS data file MYDATA.SASEMPS is used in Chapter 4, “Browsing and Updating ADABAS Data,” on page 37. It was created with the following SAS statements:

```
libname mydata 'your-SAS-library';
data mydata.sasemps;
/* employee identification */
  input empid
        /* birth date */
        birthdat date7.
        /* last name */
        lastname $18.
        /* first name */
        firstnam $15.
        /* middle name */
        middlena $15.;
  datalines;
245962 30AUG64 BEDORTHA      KATHY      MARTHA
765432 01MAR59 POWELL       FRANK      X.
219223 13JUN47 HANSINGER   BENJAMIN   HAROLD
326745 21FEB52 RAWN        BEATRICE   MAY
;
run;
```

The following PRINT procedure lists the data shown in Output 4.11:

```
proc print data=mydata.sasemps;
  title 'Data in MYDATA.SASEMPS Data File';
  format birthdat date7.;
run;
```

LIB6.BIRTHDAY Data File

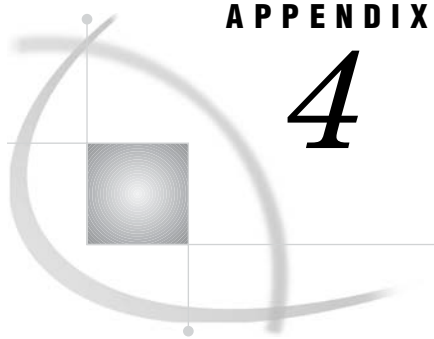
The SAS data file LIB6.BIRTHDAY is used in Chapter 3, “ADABAS Data in SAS Programs,” on page 17. It was created with the following SAS statements:

```
libname lib6 'your-SAS-library';
data lib6.birthday;
/* employee identification */
  input empid
        /* birth date */
        birthdat date7.
        /* last name */
        lastname $18.;
  datalines;
129540 31JUL60 CHOULAI
```

```
356134 25OCT60 DUNNETT
127845 25DEC43 MEDER
677890 24APR65 NISHIMATSU-LYNCH
459287 05JAN34 RODRIGUES
346917 15MAR50 SHIEKELESLAN
254896 06APR49 TAYLOR-HUNYADI
;
run;
```

The following PRINT procedure lists the data shown in Output 3.15:

```
proc print data=lib6.birthday;
  title 'LIB6.BIRTHDAY Data File';
  format birthdat date7.;
run;
```



APPENDIX

4

Recommended Reading

Recommended Reading 155

Recommended Reading

Here is the recommended reading list for this title:

- SAS Language Reference: Concepts*
- SAS Language Reference: Dictionary*
- Base SAS Procedures Guide*
- Getting Started with the SAS System in the MVS Environment*
- SAS/CONNECT User's Guide*
- SAS/GRAPH Reference, Volumes 1 and 2*
- SAS/STAT User's Guide*

For a complete list of SAS publications, see the current *SAS Publishing Catalog*. To order the most current publications or to receive a free copy of the catalog, contact a SAS representative at

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513
Telephone: (800) 727-3228*
Fax: (919) 677-8166
E-mail: sasbook@sas.com
Web address: support.sas.com/pubs

* For other SAS Institute business, call (919) 677-8000.

Customers outside the United States should contact their local SAS office.

Glossary

access descriptor

a SAS/ACCESS file that describes a single ADABAS file or NATURAL DDM to SAS. After creating an access descriptor, you can use it as the basis for creating one or more view descriptors. See also view descriptor.

Associator

an ADABAS system file that contains internal storage information that is used for managing the data for an entire database.

batch mode

a method of executing SAS programs in which a file that contains SAS statements plus any necessary operating environment commands is submitted to the computer's batch queue. After you submit the program, control returns to your terminal or workstation, where you can perform other tasks. Batch mode is sometimes referred to as running in the background. The program output can be written to files or printed on an output device.

browsing data

the process of viewing the contents of a file. You can view the data either one observation at a time or in a tabular format. You cannot update data that you are browsing.

ciphared data

data records that are stored in a form that cannot be read by non-ADABAS programs.

compressed data

ADABAS data from which blanks in alphanumeric fields and leading zeros in numeric fields have been removed. (SAS also has a compression option for SAS data files.)

data field

the smallest logical unit of information in an ADABAS file. The six types of data fields are elementary, multiple-value, group, periodic group, subfield, and superfield.

Data Storage

an ADABAS file that contains the compressed data records for all ADABAS files. A single physical block of Data Storage contains a varying number of logical records.

data value

a character value or numeric value that is stored in one variable in an observation. In other words, a data value is the intersection of a variable (the vertical component

of a data table) and an observation (the horizontal component). For example, the variable LASTNAME might contain the data value Smith.

database

an organized collection of related data. In ADABAS, a database can consist of up to 5,000 logical files. These files are contained in the Associator system file and the Data Storage file.

database management system (DBMS)

a software application that enables you to create and manipulate data that is stored in the form of databases.

descriptor (ADABAS)

an ADABAS data field that has been defined as the key data field and which provides an index of the data field's value. The SAS/ACCESS interface to ADABAS supports three descriptor types: subdescriptor, superdescriptor, and phonetic descriptor.

descriptor file

a type of SAS/ACCESS file that is used to establish a connection between SAS and ADABAS. To create descriptor files, you use the ACCESS procedure. There are two types of descriptor files: access descriptors and view descriptors.

editing data

the process of viewing the contents of a file with the intent and ability to change those contents. You can view the data either one observation at a time or as a group in a tabular format.

elementary field

an ADABAS data field that can contain only one value per record.

engine

a component of SAS software that reads from or writes to a file. Each engine enables SAS to access files that are in a particular format. See also interface view engine.

exclusive file control

the control of one or more files by a single user. Exclusive file control prevents other users from updating the file during a session.

field

See data field.

file

a collection of related records that are treated as a unit. SAS files are processed and controlled by SAS and are stored in SAS data libraries.

An ADABAS file can contain from 0 to 4,294,967,294 records. The records are physically stored in compressed form in the Data Storage file. File control information, field definitions, and inverted list entries are contained in the Associator System file.

format, variable

a pattern or set of instructions that SAS uses to determine how the values of a variable should be written or displayed. SAS provides a set of standard formats and also enables you to define your own formats.

group

several consecutive data fields that have been combined into one field so that they can be accessed more efficiently and referred to more easily.

index

a component of a SAS data set that enables SAS to access observations in the SAS data set quickly and efficiently. The purpose of SAS indexes is to optimize WHERE-clause processing and to facilitate BY-group processing.

informat, variable

a pattern or set of instructions that SAS uses to determine how data values in an input file should be interpreted. SAS provides a set of standard informats and also enables you to define your own informats.

interactive line mode

a method of running SAS programs in which you enter one line of a SAS program at a time at the SAS session prompt. SAS processes each line immediately after you press the ENTER or RETURN key. Procedure output and informative messages are returned directly to your display device.

interface view engine

a SAS engine that retrieves data directly from files that were formatted by another vendor's software. The SAS/ACCESS interface to ADABAS includes an interface view engine.

inverted list

a list that contains the different values of a descriptor data field, along with the count and the ISNs of the records that contain each value.

ISN (Internal Sequence Number)

a logical identifier for a record. Each record in an ADABAS file has a unique ISN.

key field

See descriptor (ADABAS).

libref

a temporary name that points to a SAS data library. The complete name of a SAS file consists of two words, separated by a period. The libref, which is the first word, indicates the library; the second word is the specific SAS file in the library. For example, in VLIB.NEWBDAY, VLIB is the libref that tells SAS where to look to find the file NEWBDAY.

member

a SAS file in a SAS data library.

member name

a name that is assigned to a SAS file in a SAS data library.

member type

a SAS name that identifies the type of information that is stored in a SAS file. Member types include ACCESS, DATA, CATALOG, PROGRAM, and VIEW.

missing value

a term that describes the contents of a variable that contains no data for a particular row or observation. By default, SAS prints or displays a missing numeric value as a single period, and it prints or displays a missing character value as a blank space.

multiple-value field

an ADABAS data field that can contain 0 to 191 values per record.

null value

a special value that indicates the absence of information. Null values are analogous to SAS missing values.

observation

a row in a SAS data set. Each observation contains one data value for each variable in the data file. In a database product table, an observation is analogous to a row. Unlike rows in a DBMS table or file, observations in a SAS data file have an inherent order.

occurrences

the number of values in a single record for an ADABAS multiple-value field or the number of times a periodic group is repeated in a given record.

periodic group

a collection of ADABAS data fields that can contain one or more elementary fields and multiple-value fields and which can be repeated up to 191 times.

phonetic descriptor

an ADABAS descriptor that performs a search according to a phonetic value, such as a last name. See also descriptor (ADABAS)

SAS data file

a SAS data set that contains data values as well as descriptor information that is associated with the data.

SAS data library

a collection of SAS files that are recognized by SAS and which are referenced and stored as a unit.

SAS data set

a file whose contents are in one of the native SAS file formats. There are two types of SAS data sets: SAS data files and SAS data views. SAS data files contain data values in addition to descriptor information that is associated with the data. SAS data views contain only the descriptor information plus other information that is required for retrieving data values from other SAS data sets or from files whose contents are in other software vendors' file formats.

SAS data view

one of the formats of a SAS data set that contains only the descriptor and other information required to retrieve the data values from other SAS files or external files. Both PROC SQL views and SAS/ACCESS views are considered SAS data views. SAS data views are of the member type VIEW.

subdescriptor

an ADABAS descriptor that has been derived from part of an elementary data field.

subfield

an ADABAS data field that has been defined from part of another data field. For example, the subfield AREA CODE could be the first three digits of the PHONE data field. Subfields can be used only in read operations.

superdescriptor

an ADABAS descriptor that has been derived from more than one data field, from portions of data fields, or both.

superfield

a data field that consists of several data fields, portions of fields, or both. Superfields can be used only in read operations.

system file

a type of ADABAS file that is used to store Data Definition Modules (DDMs) and NATURAL SECURITY information. System files are created and maintained by ADABAS and should not be updated directly.

variable

a column in a SAS data file. The data values for each variable describe a single characteristic for all observations (rows). Each SAS variable can have the following attributes: name, type (character or numeric), length, format, informat, and label. In the ACCESS procedure, variables are created from ADABAS data fields.

variable-length field

an ADABAS data field whose length can vary. The maximum length for a variable length field is the length that has been specified for its format.

view

a definition of a virtual data set. The definition is named and stored for later use. A view contains no data; it merely describes or defines data that is stored elsewhere. SAS data views can be created by the ACCESS and SQL procedures. See also SAS data view.

view descriptor

a SAS/ACCESS file that defines a subset of a database that is described by an access descriptor. The subset consists of selected data fields from an ADABAS file, with optional selection and ordering criteria. See also access descriptor.

Index

- A**
- ACCDESC= option
 - PROC ACCESS statement (ADABAS) 58
 - access descriptors 5
 - creating 65
 - effects of changing ADABAS files 107
 - effects of changing NATURAL DDMs 107
 - example data 147
 - invalidating 58
 - NATURAL SECURITY options 79
 - passwords 58
 - resetting column defaults 82
 - specifying 58
 - updating 87
 - ACCESS procedure, ADABAS 55, 56
 - calls on behalf of 103
 - case sensitivity 56
 - data conversion 95
 - description 57
 - formats 95
 - informats 95
 - invoking 60
 - SORT clause in view descriptors 93
 - syntax 56
 - system options 112
 - terminating 80
 - WHERE clause in view descriptors 88
 - ADABAS databases 8
 - ADABAS descriptors 10
 - ADABAS files 9
 - NATURAL data definition modules 10
 - ADABAS DBMS 8
 - ADABAS descriptors 10
 - definition 8
 - phonetic descriptors 11
 - subdescriptors 10
 - superdescriptors 10
 - ADABAS files 9
 - cipher codes 116
 - data field names 9
 - database identifiers 116
 - definition 8
 - effects of changing 107
 - example data 132
 - file numbers 9, 116
 - level numbers 9
 - logical record ISN 9
 - passwords 117
 - ADABAS interface 3
 - ACCESS procedure 55
 - database administration 102
 - effects on ADABAS data 97
 - ADABAS Security 108
 - ADBAUSE system options 112
 - ADBCC= data set option 116
 - ADBDBID= data set option 116
 - ADBDDM= data set option 116
 - ADBDEL= data set option 116
 - ADBEUSE system options 113
 - ADBFIL= data set option 116
 - ADBFIL statement
 - ACCESS procedure (ADABAS) 62
 - ADBFMTL= data set option 116
 - ADBISNL= data set option 117
 - ADBL3= data set option 116
 - ADBNATAP= data set option 117
 - ADBNATPW= data set option 117
 - ADBNATUS= data set option 117
 - ADBPW= data set option 117
 - ADBRECL= data set option 117
 - ADBSCHL= data set option 117
 - ADBSE user exit 110
 - ADBSECC= data set option 117
 - ADBSECDB= data set option 117
 - ADBSECF= data set option 117
 - ADBSECPW= data set option 117
 - ADBSYSCC= data set option 117
 - ADBSYSDB= data set option 117
 - ADBSYSFL= data set option 117
 - ADBSYSPW= data set option 117
 - ADBTRACE= data set option 112, 117
 - ADBVAL= data set option 118
 - adding ADABAS data
 - SAS/FSP procedures 41
 - ambiguous inserts 119
 - APPEND procedure
 - appending ADABAS data 49
 - ASSIGN statement
 - ACCESS procedure (ADABAS) 63
 - Associator system file 8
- B**
- browsing ADABAS data 37
 - FSBROWSE procedure 38
 - FSVIEW procedure 38
 - SAS/FSP procedures 38
 - SELECT statement 43
 - SQL procedure for 43
 - WHERE expression while browsing 39
- BY** keys
 - examples 119
 - for elementary data fields 72
 - resolving ambiguous inserts 119
 - view descriptor considerations 121
- BY** statement
 - sort order efficiency 95
- C**
- calls
 - by other SAS procedures 103
 - on behalf of ACCESS procedure 103
 - case sensitivity
 - ACCESS procedure (ADABAS) 56
 - CHART procedure
 - charting ADABAS data 20
 - cipher codes 14, 62
 - ADABAS files 116
 - DDM system files 117
 - NATURAL SECURITY system files 117
 - combining data
 - See selecting and combining data
 - competitive updating 107
 - CONTENT statement
 - ACCESS procedure (ADABAS) 64
 - CREATE statement
 - ACCESS procedure (ADABAS) 65
- D**
- data conversion
 - ACCESS procedure (ADABAS) 95
 - data definition module
 - See DDM (data definition module)
 - data field names 9
 - DDMs 10
 - data fields 11
 - definition 8
 - elementary fields 11
 - formats 13
 - group fields 11
 - length of 13
 - mapping data 12
 - multiple-value fields 11

- periodic group fields 12
 - subfields 12
 - superfields 12
 - types of 11
- data locks 111
- data redundancy 118
- data security
 - See* security
- data set options 116
- Data Storage system file 8
- database administration 101
 - ACCESS procedure, system options 112
 - ADABAS interface 102
 - competitive updating 107
 - data locks 111
 - data security 108
 - debugging 112
 - effects of changing ADABAS files 107
 - effects of changing NATURAL DDMs 107
 - interface view engine 102
 - interface view engine, system options 112
 - logical transaction recovery 107
 - performance 111
 - retrieval processing 103
 - update processing 106
- database identifiers 8, 62
 - ADABAS files 116
 - DDM system files 117
 - NATURAL SECURITY system files 117
- database names 8
- databases
 - See* ADABAS databases
- DATASETS procedure
 - assigning passwords 59
- date formats 64
- DBMS (ADABAS) 8
- DBMS columns
 - changing formats 71
 - changing informats 71
 - data field information 74
 - dropping 69
 - listing, with information 69, 73
 - resetting to defaults 81
 - selecting 84
- DBMS= option
 - PROC ACCESS statement (ADABAS) 58
- DBMS tables
 - invalidating descriptors 58
- DDM (data definition module) 10
 - data field names 10
 - effects of changing 107
 - example data 145
 - filename 10
 - name specification 68, 116
 - system file containing 86
- DDM= statement
 - ACCESS procedure (ADABAS) 68
- DDM system files 117
- debugging 112
- DELETE statement
 - deleting ADABAS data 47
 - WHERE clause and 48
- deleting ADABAS data 47
 - SAS/FSP procedures 41
- descriptor files 4, 8
 - access descriptor files 5
 - creating 65

- effects of changing ADABAS files 107
- effects of changing NATURAL DDMs 107
- updating 87
- view descriptor files 5

DROP statement

- ACCESS procedure (ADABAS) 69

E

- elementary fields 11
 - BY key for 72
- example data 5, 132
 - access descriptors 147
 - ADABAS files 132
 - DDMs 145
 - SAS data files 152
 - view descriptors 150
- expressions
 - in view WHERE clauses 90
- EXTEND statement
 - ACCESS procedure (ADABAS) 69
- extended time formats 64
- extracting ADABAS data 34, 95

F

- file numbers 9, 62
 - ADABAS files 116
 - DDM system file 117
 - NATURAL SECURITY system file 117
- file objects 15
- files
 - See* ADABAS files
- FORMAT statement
 - ACCESS procedure (ADABAS) 71
- formats
 - ACCESS procedure (ADABAS) 95
 - automatically generated 63
- FREQ procedure 22
- FSBROWSE procedure
 - browsing ADABAS data 38
- FSEDIT procedure
 - updating ADABAS data 38
- FSVIEW procedure
 - browsing and updating ADABAS data 38

G

- GROUP BY clause
 - creating new variables 31
- group fields 11

H

- hyperdescriptors 11

I

- INFORMAT statement
 - ACCESS procedure (ADABAS) 71

- informats
 - ACCESS procedure (ADABAS) 95
- INSERT statement
 - inserting ADABAS data 47
- inserting ADABAS data 47
 - ambiguous inserts 119
- interface to ADABAS
 - See* ADABAS interface
- interface view engine 4, 102
 - calls 103
 - mapping ADABAS data into observations 97
 - overriding default values 116
 - security and 108
 - system options 112
- internal sequence number (ISN) 9, 102
- inverted lists 10
- ISN (internal sequence number) 9, 102

K

- KEEP= data set option 25
- KEY statement
 - ACCESS procedure (ADABAS) 72

L

- level numbers 9
- library identifier 117
- library objects 15
- links
 - NATURAL SECURITY and 15
- LIST statement
 - ACCESS procedure (ADABAS) 73
- LISTINFO statement
 - ACCESS procedure, ADABAS 74
- LISTOCC statement
 - ACCESS procedure (ADABAS) 75
- logical record ISN 9
- logical records 8
- logical transaction recovery 107

M

- mapping data 12, 97
- MEANS procedure 22
- missing values 121
- multiple-value fields 11
 - DB content attribute 77
 - dropping 77
 - format attribute 77
 - in selection criteria 122
 - information attribute 77
 - listing occurrences of 75
 - mapping to SAS observations 97
 - modifying occurrences of 76
 - occurrences 78
 - renaming variables 78
 - resetting attributes 78
 - SAS variables and 98
 - selecting occurrences 78
 - view WHERE clauses and 93
- MVF statement
 - ACCESS procedure (ADABAS) 76

N

- NATURAL 8
 - See also* DDM (data definition module)
 - file objects 15
 - library objects 15
 - user objects 15
- NATURAL SECURITY 108
 - library identifier 117
 - links and 15
 - options 15, 79
 - password specification 117
 - system file parameters 83
 - system files 117
 - user identifier 117
- NSS statement
 - ACCESS procedure (ADABAS) 79
- null values 14, 121

O

- observations
 - adding 118
 - deleting 118
 - mapping ADABAS data into 97
 - null values 121
- OUT= option
 - PROC ACCESS statement (ADABAS) 58

P

- passwords 14, 62
 - access descriptors 58
 - ADABAS files 117
 - assigning 59
 - DDM system files 117
 - NATURAL SECURITY 117
 - view descriptors 58
- percentages 22
- performance 34, 111
- periodic group fields 12
 - adding 119
 - in selection criteria 124
 - mapping to SAS observations 98
 - SAS variables and 98
 - view WHERE clauses and 93
- permanent WHERE clauses 39
- phonetic descriptor fields 98
- phonetic descriptors 11
- PRINT procedure
 - printing ADABAS data 19
- PROC ACCESS statement (ADABAS) 58

Q

- QUIT statement
 - ACCESS procedure (ADABAS) 80

R

- RANK procedure 24
- redundancy 118

- RENAME statement
 - ACCESS procedure (ADABAS) 80
- renaming variables 80
 - multiple-value fields 78
- RESET statement
 - ACCESS procedure (ADABAS) 81
- retrieval processing 103
 - no sorting criteria 104
 - no WHERE clause 104
 - sorting criteria with 106
 - with only a WHERE clause 105

S

- SAS/ACCESS descriptor files 8
- SAS/ACCESS interface to ADABAS
 - See* ADABAS interface
- SAS/FSP procedures
 - adding and deleting ADABAS data 41
 - browsing and updating ADABAS data 38
- SAS security 110
- SECFILE statement
 - ACCESS procedure (ADABAS) 83
 - security 108
 - See also* NATURAL SECURITY
 - ADABAS options 14
 - ADABAS Security 108
 - ADBSE user exit 110
 - database administration 108
 - effects of changing security options 110
 - interface view engine and 108
 - NATURAL options 15
 - SAS security 110
 - specifications 84
- SECURITY statement
 - ACCESS procedure (ADABAS) 84
- SELECT statement
 - ACCESS procedure (ADABAS) 84
 - browsing and updating ADABAS data 43
 - selecting and combining data 24
 - SQL procedure 26
 - WHERE statement 25
- selection criteria
 - multiple-value fields in 122
 - periodic group fields in 124
 - specifying 130
 - view descriptors 85, 94
 - view WHERE clause for 130
 - WHERE clauses 127, 130
- SORT clauses
 - in view descriptors 93
- sort order efficiency 95
- SORT procedure
 - ADABAS 95
- sorting criteria
 - omitting, in retrieval processing 104
 - retrievals with 106
- SPANS operator
 - in view WHERE clauses 90
- SQL procedure
 - browsing and updating ADABAS data 43
 - combining data from various sources 26
 - creating variables with GROUP BY clause 31
 - selecting and combining data 26
 - statistics calculation 22
- subdescriptor fields 92, 98

- subdescriptors 10
- subfields 12, 96
- SUBSET statement
 - ACCESS procedure (ADABAS) 85
- superdescriptor fields 91
 - data conversion 96
 - SAS variables and 98
- superdescriptors 10
- superfields 12
- SYSPFILE statement
 - ACCESS procedure (ADABAS) 86
- system files 8
 - Associator system file 8
 - containing DDMs 86
 - Data Storage system file 8
 - DDM 117
 - NATURAL SECURITY 83, 117
 - Work Storage system file 8
- system options
 - ACCESS procedure (ADABAS) 112
 - interface view engine 112

T

- temporary WHERE clauses 39
- trace options 117
- transaction data 32

U

- update processing 106
- UPDATE statement
 - ACCESS procedure (ADABAS) 87
 - updating ADABAS data 45
- updating, competitive 107
- updating access descriptors 87
- updating ADABAS data 37
 - FSEDIT procedure 38
 - FSVIEW procedure 38
 - SAS/FSP procedures for 38
 - SELECT statement 43
 - SQL procedure 43
- UPDATE statement 45
 - WHERE expression while updating 39
- updating data files 32
- updating view descriptors 87
- user exits
 - ADBSE 110
- user identifier 117
- user objects 15

V

- variable length 64
- variable length fields 13
- variable names
 - automatically generated 63
- variables
 - creating with GROUP BY clause 31
 - renaming 78, 80
 - reviewing ADABAS variables 18
- view descriptors 5
 - BY key considerations 121

- creating 65, 66
- DBMS columns, selecting 84
- effects of changing ADABAS files 107
- effects of changing NATURAL DDMs 107
- efficient use of 94
- example data 150
- invalidating 58
- multiple 118
- passwords 58
- performance 34
- resetting column defaults 82
- selection criteria 85, 94
- SORT clauses in 93
- specifying 58
- updating 87
- WHERE clauses in 88
- view SORT clauses 93
 - examples 94
 - syntax 93
- view WHERE clauses 88
 - character field values 90

- dates 91
- examples 89
- expressions 90
- multiple-value field values 93
- numeric format values 90
- periodic group field values 93
- selection criteria 130
- SPANS operator 90
- subdescriptor field values 92
- superdescriptor field values 91
- syntax 88
- VIEWDESC= option
 - PROC ACCESS statement (ADABAS) 58
- views 5

W

- WHERE clause processing
 - ADABAS 95
- WHERE clauses
 - acceptable conditions 128
 - DELETE statement and 48
 - in retrievals 105
 - in view descriptors 88
 - multiple-value fields in selection criteria 123
 - omitting, in retrieval processing 104
 - periodic group fields in selection criteria 125
 - permanent 39
 - referencing descriptor data fields 129
 - selection criteria 127, 130
 - temporary 39
 - unacceptable conditions 129
- WHERE expressions
 - while browsing/updating data 39
- WHERE statement
 - selecting and combining data 25
- Work Storage system file 8

Your Turn

If you have comments or suggestions about *SAS/ACCESS® 9.1 Interface to ADABAS: Reference*, please send them to us on a photocopy of this page or send us electronic mail. Send comments about this document to, please return the photocopy to

SAS Publishing
SAS Campus Drive
Cary, NC 27513
email: yourturn@sas.com

For suggestions about the software, please return the photocopy to

SAS Institute Inc.
Technical Support Division
SAS Campus Drive
Cary, NC 27513
email: suggest@sas.com

