



THE  
POWER  
TO KNOW.

# **SAS/ACCESS<sup>®</sup> 9.2**

## **Interface to CA-Datcom/DB**

### **Reference**

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2008. *SAS/ACCESS® 9.2 Interface to CA-Datcom/DB: Reference*. Cary, NC: SAS Institute Inc.

**SAS/ACCESS® 9.2 Interface to CA-Datcom/DB: Reference**

Copyright © 2008, SAS Institute Inc., Cary, NC, USA

ISBN 978-1-59047-933-9

All rights reserved. Produced in the United States of America.

**For a hard-copy book:** 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.

**For a Web download or e-book:** Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

**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 electronic book, March 2008

1st printing, March 2008

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/publishing](http://support.sas.com/publishing) 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

<b>PART 1</b>	<b>SAS/ACCESS Interface to CA-Datcom/DB: Usage</b>	<b>1</b>
	<b>Chapter 1</b> △ <b>Overview of the SAS/ACCESS Interface to CA-Datcom/DB</b>	<b>3</b>
	Introduction to the SAS/ACCESS Interface to CA-Datcom/DB	3
	Purpose of the SAS/ACCESS Interface to CA-Datcom/DB	3
	SAS/ACCESS Descriptor Files	4
	Example Data in the CA-Datcom/DB Document	6
	<b>Chapter 2</b> △ <b>CA-Datcom/DB Essentials</b>	<b>7</b>
	Introduction to CA-Datcom/DB Essentials	7
	CA-Datcom/DB and CA-DATADICIONARY Software	7
	CA-Datcom/DB Databases	9
	CA-Datcom/DB Indexing	12
	Selecting a Subset of CA-Datcom/DB Data	12
	Sorting Data in a SAS/ACCESS View Descriptor	12
	Security Features for CA-Datcom/DB	13
	CA-Datcom/DB Execution Environments	13
	<b>Chapter 3</b> △ <b>Defining SAS/ACCESS Descriptor Files</b>	<b>15</b>
	Introduction to Defining SAS/ACCESS Descriptor Files	15
	SAS/ACCESS Descriptor Files Essentials	15
	Creating SAS/ACCESS Descriptor Files	16
	Updating Descriptor Files	20
	Extracting CA-Datcom/DB Data with the ACCESS Procedure	21
	<b>Chapter 4</b> △ <b>Using CA-Datcom/DB Data in SAS Programs</b>	<b>23</b>
	Introduction to Using CA-Datcom/DB Data in SAS Programs	23
	Reviewing Columns for CA-Datcom/DB Data	24
	Printing CA-Datcom/DB Data	25
	Charting CA-Datcom/DB Data	26
	Calculating Statistics for CA-Datcom/DB Data	28
	Selecting and Combining CA-Datcom/DB Data	30
	Updating a SAS Data File with CA-Datcom/DB Data	38
	Performance Considerations	41
	<b>Chapter 5</b> △ <b>Browsing and Updating CA-Datcom/DB Data</b>	<b>43</b>
	Introduction to Browsing and Updating CA-Datcom/DB Data	43
	Browsing and Updating CA-Datcom/DB Data with the SAS/FSP Procedures	44
	Browsing and Updating CA-Datcom/DB Data with the SQL Procedure	50
	Appending CA-Datcom/DB Data with the APPEND Procedure	53
<b>PART 2</b>	<b>SAS/ACCESS Interface to CA-Datcom/DB: Reference</b>	<b>57</b>

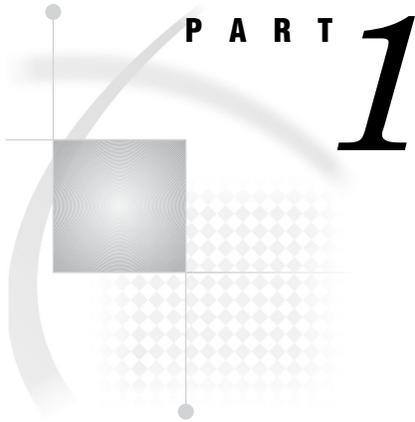
<b>Chapter 6</b>	<b>△ ACCESS Procedure Reference</b>	<b>59</b>
Introduction to ACCESS Procedure Reference		59
ACCESS Procedure Syntax		60
SAS Passwords for SAS/ACCESS Descriptors		63
Invoking the ACCESS Procedure		65
Statements		67
WHERE Clause in a View Descriptor		89
SORT Clause in a View Descriptor		95
Creating and Using View Descriptors Efficiently		96
ACCESS Procedure Data Conversions		97

## **PART 3**   **Appendixes**   **101**

<b>Appendix 1</b>	<b>△ Information for the Database Administrator</b>	<b>103</b>
Introduction to the Information for the Database Administrator		103
How the SAS/ACCESS Interface to CA-Datcom/DB Works		104
Retrieval Processing		105
Update Processing		107
Recovery Processing		108
How Changing the CA-DATADictionary Database Affects Descriptor Files		109
SAS Security with CA-Datcom/DB		110
User Requirements Table (URT)		110
Locks and the Spool Files		111
Direct Addressing and Access by Row Number		111
Password Encryption/Decryption in CA-Datcom/DB		112
Maximizing the CA-Datcom/DB Interface Performance		112
Multi-Tasking with CA-Datcom/DB		112
Error Messages and Debugging Information for CA-Datcom/DB		113
System Options for the CA-Datcom/DB Interface		113
<b>Appendix 2</b>	<b>△ Advanced Topics</b>	<b>117</b>
Introduction to Advanced Topics		117
Data Set Options		117
Using Multiple View Descriptors		124
User Exits from CA-Datcom/DB		124
Deleting and Inserting Data Records in CA-Datcom/DB		124
Missing Values (Nils) in CA-Datcom/DB Tables		125
SAS WHERE Clause Conditions Not Acceptable to CA-Datcom/DB		125
Deciding How to Specify Selection Criteria in CA-Datcom/DB		126
Validation of Data Values in CA-Datcom/DB		126
Validation against CA-DATADictionary		126
<b>Appendix 3</b>	<b>△ Data and Descriptors for the Examples</b>	<b>129</b>
Introduction to Data and Descriptors for the Examples		129
CA-Datcom/DB Tables		130
Access Descriptors for the CA-Datcom/DB Tables		144

View Descriptors for the CA-Datacom/DB Tables	146
SAS Data Files Used for CA-Datacom/DB Examples	149
<b>Appendix 4 <math>\triangle</math> Recommended Reading</b>	<b>153</b>
Recommended Reading	153
<b>Glossary</b>	<b>155</b>
<b>Index</b>	<b>161</b>

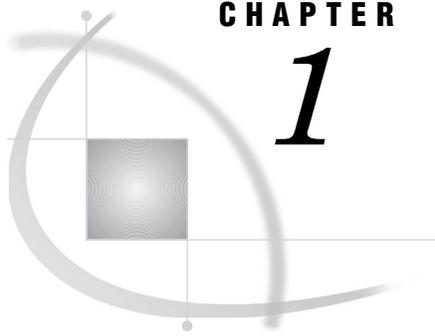




## **SAS/ACCESS Interface to CA-Datcom/DB: Usage**

<i>Chapter 1</i> . . . . .	<b>Overview of the SAS/ACCESS Interface to CA-Datcom/DB</b>	<b>3</b>
<i>Chapter 2</i> . . . . .	<b>CA-Datcom/DB Essentials</b>	<b>7</b>
<i>Chapter 3</i> . . . . .	<b>Defining SAS/ACCESS Descriptor Files</b>	<b>15</b>
<i>Chapter 4</i> . . . . .	<b>Using CA-Datcom/DB Data in SAS Programs</b>	<b>23</b>
<i>Chapter 5</i> . . . . .	<b>Browsing and Updating CA-Datcom/DB Data</b>	<b>43</b>





## CHAPTER

## 1

## Overview of the SAS/ACCESS Interface to CA-Datcom/DB

<i>Introduction to the SAS/ACCESS Interface to CA-Datcom/DB</i>	3
<i>Purpose of the SAS/ACCESS Interface to CA-Datcom/DB</i>	3
<i>SAS/ACCESS Descriptor Files</i>	4
<i>Access Descriptor and View Descriptor Files</i>	4
<i>Access Descriptor Files</i>	5
<i>View Descriptor Files</i>	5
<i>Example Data in the CA-Datcom/DB Document</i>	6

### Introduction to the SAS/ACCESS Interface to CA-Datcom/DB

This section introduces you to SAS/ACCESS software and briefly describes how to use the interface. This section also introduces the sample CA-Datcom/DB tables, view descriptors, and SAS data files used in this document.

### Purpose of the SAS/ACCESS Interface to CA-Datcom/DB

SAS/ACCESS software provides an interface between SAS and the CA-Datcom/DB *database management system* (DBMS). You can perform the following tasks with this SAS/ACCESS interface:

- Create SAS/ACCESS descriptor files using the ACCESS procedure.
- Directly access data in CA-Datcom/DB tables from within a SAS program, using the descriptor files created with the ACCESS procedure.
- Extract CA-Datcom/DB data and place it in a SAS data file using the ACCESS procedure or the DATA step.
- Update data in CA-Datcom/DB tables using the SQL procedure, the APPEND procedure, SAS/FSP software, or SAS/AF software.

This 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 CA-Datcom/DB data in SAS programs in much the same way as you use SAS data files

The ACCESS procedure enables you to describe CA-Datcom/DB 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 files, and so on. Chapter 4, “Using CA-Datcom/DB Data in SAS Programs,” on page 23 presents

several examples of using CA-Datcom/DB data in SAS programs. Chapter 5, “Browsing and Updating CA-Datcom/DB Data,” on page 43 shows you how to use descriptor files to update CA-Datcom/DB data from within a SAS program.

The interface view engine is an integral part of the SAS/ACCESS interface, but you do not have to deal directly with the engine. SAS automatically interacts with the engine when you use view descriptors in your SAS programs, so you can simply use CA-Datcom/DB data just as you would use SAS data.

You can combine data from several CA-Datcom/DB tables. Such combinations are not only possible but easy to do. SAS can distinguish among SAS data files, SAS/ACCESS descriptor files, and other types of SAS files, and the software will use the appropriate access method. See “Updating a SAS Data File with CA-Datcom/DB Data” on page 38 for an example.

---

## SAS/ACCESS Descriptor Files

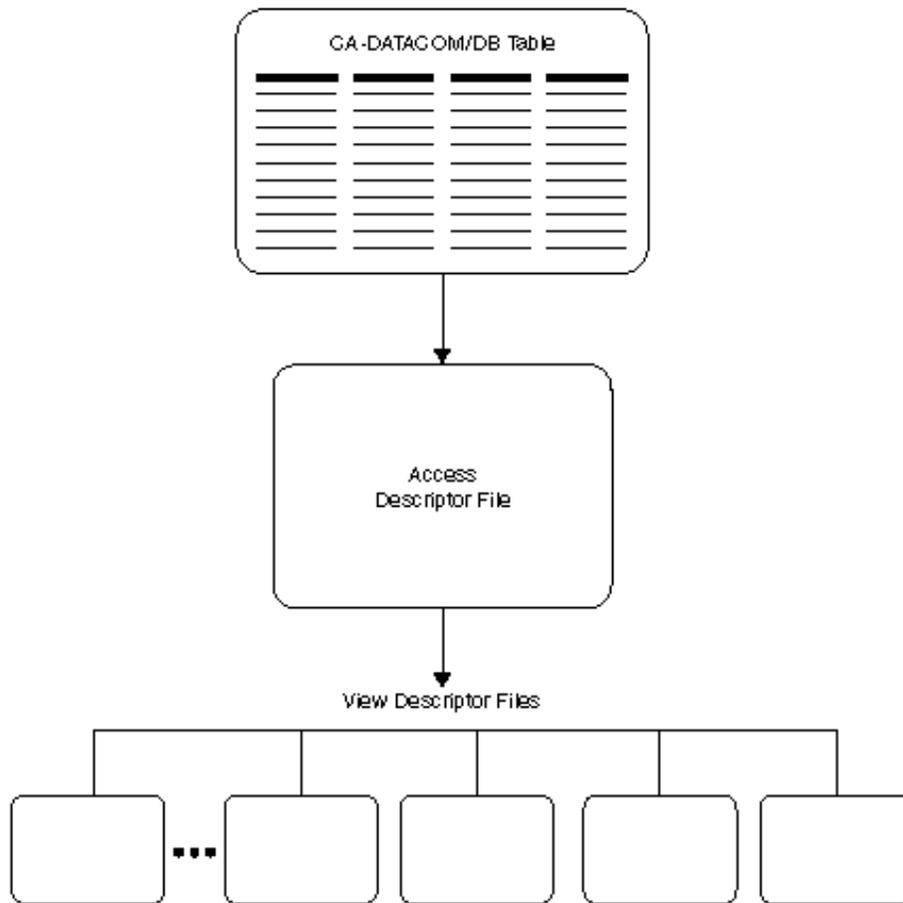
---

### Access Descriptor and View Descriptor Files

SAS/ACCESS software uses SAS/ACCESS descriptor files to establish a connection between SAS and CA-Datcom/DB. To create these files, you use the ACCESS procedure.

There are two types of descriptor files: *access descriptors* and *view descriptors*. The following figure illustrates the relationships among a CA-Datcom/DB table, access descriptors, and view descriptors.

**Figure 1.1** Relationships between a CA-Datcom/DB Table, Access Descriptor Files, and View Descriptor Files



The two types of descriptor files are discussed next. Chapter 3, “Defining SAS/ACCESS Descriptor Files,” on page 15 shows you how to create and edit these files.

---

## Access Descriptor Files

Access descriptor files are of member type ACCESS. Each access descriptor holds essential information about one CA-Datcom/DB table you want to access, for example, the table name, field names, and data types. It also contains corresponding SAS information, such as the SAS column names, formats, and informats. Typically, you have one access descriptor for each CA-Datcom/DB table.

An access descriptor describes one CA-Datcom/DB table. You cannot create a single access descriptor that references two CA-Datcom/DB tables.

---

## View Descriptor Files

View descriptor files are sometimes called SAS *views*, because their member type is VIEW. This document uses the term *view descriptor* to distinguish them from views created by the 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 CA-Datcom/DB table). For

example, you might want to use only three of four possible fields in the table and only some of the values stored in the fields. The view descriptor enables you to select the 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 January 3, 1998, and that customers' names begin with W.) Typically, for each access descriptor, you have several view descriptors, which select different subsets of data.

You can join data from multiple CA-Datcom/DB tables with the SQL procedure. The SQL procedure can join SAS data files, PROC SQL views, and SAS/ACCESS view descriptors. See Chapter 4, "Using CA-Datcom/DB Data in SAS Programs," on page 23 and Chapter 5, "Browsing and Updating CA-Datcom/DB Data," on page 43 for examples that use the SQL procedure.

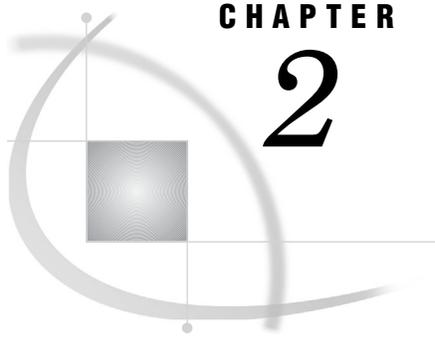
---

## Example Data in the CA-Datcom/DB Document

This document uses several CA-Datcom/DB tables to show you how to use the SAS/ACCESS interface to CA-Datcom/DB. The tables were created for an international textile manufacturer. This company's product line includes some special fabrics that are made to precise specifications. The tables are named CUSTOMERS, EMPLOYEES, INVOICE, and ORDER. All the data is fictitious.

These tables are designed to show how the interface treats CA-Datcom/DB data. They are not meant as examples for you to follow in designing tables for any purpose.

Appendix 3, "Data and Descriptors for the Examples," on page 129 gives more information about the tables, the data each table contains, and the sample descriptors. Appendix 3 also gives information about the sample SAS data files used in Chapter 4, "Using CA-Datcom/DB Data in SAS Programs," on page 23.



## CA-Datcom/DB Essentials

---

<i>Introduction to CA-Datcom/DB Essentials</i>	7
<i>CA-Datcom/DB and CA-DATADictionary Software</i>	7
<i>CA-Datcom/DB Databases</i>	9
<i>Overview of CA-Datcom/DB Databases</i>	9
<i>DATABASE Entity-type</i>	9
<i>RECORD Entity-type</i>	9
<i>FIELD Entity-type</i>	10
<i>Data Types in CA-Datcom/DB Fields</i>	10
<i>Numeric Data Types</i>	10
<i>Character Data Types</i>	11
<i>Date Types</i>	11
<i>Missing Values</i>	11
<i>CA-Datcom/DB Indexing</i>	12
<i>Selecting a Subset of CA-Datcom/DB Data</i>	12
<i>Sorting Data in a SAS/ACCESS View Descriptor</i>	12
<i>Security Features for CA-Datcom/DB</i>	13
<i>CA-Datcom/DB Execution Environments</i>	13

---

### Introduction to CA-Datcom/DB Essentials

This section introduces SAS users to CA-Datcom/DB. It focuses on the terms and concepts that will help you use the SAS/ACCESS interface to CA-Datcom/DB. It includes descriptions of the following:

- CA-Datcom/DB and CA-DATADictionary software
- CA-Datcom/DB databases, tables, records, and fields
- CA-Datcom/DB data types
- CA-Datcom/DB indexing
- CA-Datcom/DB security features

If you want more information than this section provides, see the appropriate CA-Datcom/DB documentation. For more information about CA-Datcom/DB considerations, see Appendix 1, “Information for the Database Administrator,” on page 103 and Appendix 2, “Advanced Topics,” on page 117.

---

### CA-Datcom/DB and CA-DATADictionary Software

CA-Datcom/DB is a *database management system* (DBMS). The databases are fully defined with CA-DATADictionary.

A CA-Datcom/DB database consists of various entity-types, which can occur one or more times. For example, a database has areas, files, and records. Each data record in a table has one or more fields. The order of the data records is determined by the value for the field specified as the Native Key. Each field contains one type of data, and each record can hold one data value for each field, except that a repeating field can assume many values.

*Note:* CA-Datcom/DB views are not supported by the SAS/ACCESS interface to CA-Datcom/DB.  $\Delta$

CA-DATADictionary is a central, integrated, and active control facility that provides the basis for shared and consistent system resource management. As a repository for descriptive data, CA-DATADictionary is your tool for the following tasks:

- managing definitions and syntax
- enforcing naming conventions
- creating data relationships
- managing test and production environments

CA-DATADictionary enables you to collect information in categories called entity-types. Any data you enter into CA-DATADictionary is associated with a category, that is, an entity. For example, DATABASE, AREA, and FIELD are some specific CA-DATADictionary entities.

Each instance within the entity is an entity-occurrence. For example, defining a database involves storing information about the database in the DATABASE occurrence. Each database is listed by its unique name as an occurrence of the DATABASE entity.

Each occurrence has specific attributes such as data type. These attributes enable you to describe specific properties of each occurrence. For example, you can specify the type of data that a field contains or whether a key is a Master Key or a Native Key. The actual information you store for each attribute is an attribute value. In addition, CA-DATADictionary enables you to define support data, such as aliases, CA-Datcom/DB descriptors, text, and relationships.

CA-DATADictionary enables you to have many copies (versions) of the same occurrence. Each version of an occurrence can have one status at a given time. Only TEST and PROD are used with the SAS/ACCESS interface to CA-Datcom/DB. The five status values are as follows:

- TEST
- PROD (for production)
- HIST (for history)
- INCO (for incomplete)
- QUAL (for qualified production)

Depending on your site, you might find that using CA-DATADictionary interactively is more efficient for some tasks, while other tasks are simpler with batch jobs.

For more information about CA-DATADictionary and other CA-Datcom/DB features or administration of CA-Datcom/DB databases, see the appropriate CA-Datcom/DB documentation.

---

## CA-Datcom/DB Databases

---

### Overview of CA-Datcom/DB Databases

A CA-Datcom/DB database is a collection of CA-Datcom/DB tables, organized within certain CA-Datcom/DB areas and files. Each table consists of records that have one or more FIELD entity-occurrences.

Typically, a database is organized according to the types of data and how you want to use the data. You must understand and be familiar with your database's organization in order to retrieve and update information accurately and efficiently. And you must be familiar with the organization and contents of the database to create descriptor files for the SAS/ACCESS interface.

You need to know about several CA-Datcom/DB entity-types to use the SAS/ACCESS interface to CA-Datcom/DB. The most important entity-types are databases, records, and fields. Fields contain the actual data values, which are either character or numeric type.

You can define a field as a simple field or a compound field. Fields can also become keys. Two special keys, the Native Key and the Master Key, are required for each table. CA-Datcom/DB generates an index for each key field. Knowing about the Native Key and the indexes can help you minimize CA-Datcom/DB processing time for your view descriptors. In addition, fields can repeat. For more information about fields, see "FIELD Entity-type" on page 10.

The following sections describe the various CA-Datcom/DB entity-types that pertain to the SAS/ACCESS interface to CA-Datcom/DB.

---

### DATABASE Entity-type

Each DATABASE entity-occurrence in the CA-DATADITIONARY database has a unique name, from 1 to 32 characters long. A database also has a status (TEST or PROD) and version associated with it.

---

### RECORD Entity-type

A *table* consists of some number of records, each having one or more fields. The table name is the name of a RECORD entity-occurrence, up to 32 characters long. Data records in the table are ordered by the values for an assigned field called the Native Key. CA-Datcom/DB permits up to 240 tables in a database. The tables can be spread across one or more CA-Datcom/DB areas. When you define a record for a table, you must define at least one field, one key, and one element for that record.

To create descriptor files for the SAS/ACCESS interface, you must know the name of the RECORD entity-occurrence (table) and the user ID and optional password for the CA-DATADITIONARY. An access descriptor and its associated view descriptors pertain to only one table.

Output 2.1 illustrates four fields from the table CUSTOMERS. Field names are shown at the top of the columns. Each row represents the values in a record. The first field, CUSTOMER, is the Native Key in this table, which causes the records to be maintained in order by customer number.

**Output 2.1** A Sample CA-Datcom/DB Table

CUSTOMER	CITY	STATE	COUNTRY
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

---

## FIELD Entity-type

Each FIELD entity-occurrence has a name (of up to 32 characters) and specific attributes, such as the data type. For more information about data types, see “Data Types in CA-Datcom/DB Fields” on page 10.

You can define several kinds of fields, as described briefly here.

- A *simple field* is a single field.
- A *compound field* consists of two or more simple or compound fields. The fields can be of different data types and lengths; they can also repeat or be within repeating fields. The fields making up a compound field must be contiguous.
- A *key field* enables you to quickly select and sequence data records. A key field can be any combination of simple and compound fields, up to 180 characters. The fields in a key do not have to be contiguous.
- The *Native Key* is the field that determines the order of the records in a CA-Datcom/DB table. Each table must have one Native Key. It can be the same as the Master Key.
- The *Master Key* enables you to prevent duplicate values in a key field and to prevent changing values in that key. Each record must have one Master Key. It can be the same as the Native Key.
- A *repeating field* is a simple field or a compound field that can occur more than once. Repeating fields can also be nested within other repeating fields.
- An *element* is a unit of transfer between application programs and CA-Datcom/DB. It consists of one or more contiguous fields. An element should contain only those fields that an application program uses at execution time. When defining an element, group together fields that are frequently accessed together in applications.

---

## Data Types in CA-Datcom/DB Fields

A CA-Datcom/DB field can be any one of a variety of data types; they are mostly type character or type numeric, as discussed below.

When you create a view descriptor, the ACCESS procedure assigns SAS formats, informats, and so on, in addition to SAS column names from the CA-Datcom/DB field names. See “ACCESS Procedure Data Conversions” on page 97 for the default SAS column formats and informats for each CA-Datcom/DB data type. You can change the default formats and informats.

---

## Numeric Data Types

Here are some of the numeric types available for CA-Datcom/DB fields:

B	binary
D	packed decimal
E	extended floating-point
L	long floating-point
N	numeric (zoned decimal)
S	short floating-point
2	halfword binary (aligned)
4	fullword binary (aligned)
8	doubleword binary (aligned).

---

## Character Data Types

Here are some of the character types available for CA-Datcom/DB fields:

C	character
G	graphics data
H	hexadecimal character
K	kanji (same as type Y)
T	PL/I bit representation
Y	double-byte character set (DBCS)
Z	mixed DBCS and single byte.

---

## Date Types

CA-Datcom/DB supports the CA-Datcom/DB SQL types SQL-DATE, SQL-TIME, and SQL-STMP as binary data.

### SQL-DATE

specifies date values in the format CCYYMMDD, where CC=century, YY=year, MM=month, and DD=day.

### SQL-TIME

specifies the time values in the format HHMMSS, where HH=hours, MM=minutes, and SS=seconds.

### SQL-STMP

specifies a date and a time and adds microseconds in the format CCYYMMDDHHMMSSNNNNNN.

See “ACCESS Procedure Data Conversions” on page 97 for information about the default formats that the ACCESS procedure assigns to the DBMS data types. To specify a different representation, you can change the default SAS format in your descriptor files.

---

## Missing Values

Missing values in a CA-Datcom/DB table are referred to as *nil values* or simply *nils*. Nil values for both character and numeric type data are blanks, that is, HEX (40)s. All

fields of a key must contain blanks for a value to be nil. There are no valid packed decimal or zoned decimal nil values. You can specify binary zeros for nils (see “System Options for the CA-Datcom/DB Interface” on page 113).

In SAS, nils are referred to as *missing values*. CA-Datcom/DB and SAS handle missing values differently, but the interface view engine takes care of the differences. See “Missing Values (Nils) in CA-Datcom/DB Tables” on page 125 for a discussion of the differences.

---

## CA-Datcom/DB Indexing

An index area is required for each CA-Datcom/DB database. CA-Datcom/DB creates an index entry for each key value in each record. The indexes enable you to retrieve records quickly based on the record’s contents.

---

## Selecting a Subset of CA-Datcom/DB Data

A database would not be very efficient if all records had to be accessed when you needed data from only some of them. Therefore, you can specify selection criteria to identify those parts of the CA-Datcom/DB table that you want to access.

Selection criteria contain one or more conditions that values must meet. Typically, a condition consists of a field name, an operator, and a value, but you can also compare the values of two fields or give a range of values. Conditions can be combined with AND (&) or OR (|).

Here are some sample conditions.

```
cost<.50
lastname eq 'Smith'
```

```
part=9567 & onhand>2.0e+6
```

For the SAS/ACCESS interface to CA-Datcom/DB, you can include a WHERE clause in a view descriptor to specify selection criteria or you can include a SAS WHERE clause in a SAS program. Or you can include both WHERE clauses. The interface view engine translates WHERE clauses into CA-Datcom/DB selection criteria.

Note that the WHERE clause for a view descriptor and the SAS WHERE clause have some differences. For more information about WHERE clauses and a description of the syntax, see “WHERE Clause in a View Descriptor” on page 89 and “Deciding How to Specify Selection Criteria in CA-Datcom/DB” on page 126.

---

## Sorting Data in a SAS/ACCESS View Descriptor

Records in a CA-Datcom/DB table are maintained in order by values in the specified Native Key. In a SAS/ACCESS view descriptor, you can provide a different Default Key for the view, and the records will then assume the order of your specified Default Key. You can also specify a *SORT clause*, which consists of the keyword SORT followed by one or more field names, separated by commas. You can specify ascending or descending order for each sort key; the default is ascending order. Here is an example:

```
sort state, city, lastname desc
```

In addition, you can specify data order in a SAS program using a SAS BY clause. Note, however, that a SAS BY clause overrides a SORT clause stored in a view descriptor unless the SAS procedure includes the NOTSORTED option. In this situation, the SAS BY clause is ignored, and the SORT clause in the view descriptor is used.

For more information about SORT clauses, see “SORT Clause in a View Descriptor” on page 95.

---

## Security Features for CA-Datacom/DB

The CA-DATADictionary database is protected by user IDs, passwords, and locks. You must give the correct user ID and optional password to the ACCESS procedure so the procedure can obtain CA-DATADictionary information for creating an access descriptor. CA-Datacom/DB also has security interfaces to packages such as RACF. In addition, you can develop your own security program through a user exit in the interface view engine.

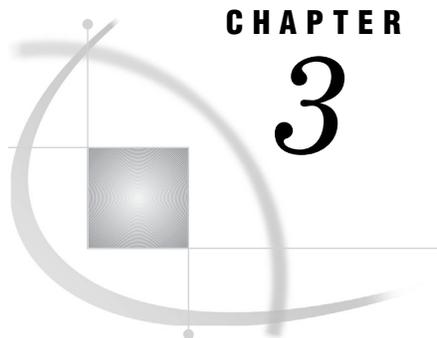
---

## CA-Datacom/DB Execution Environments

When you access a CA-Datacom/DB database, you can work in either a single user execution environment or a multi-user execution environment. In a *single user environment*, each user has a copy of CA-Datacom/DB and has exclusive use of the database.

The SAS/ACCESS interface to CA-Datacom/DB runs only under a *multi-user environment*. In a multi-user environment, many databases can be accessed from many regions concurrently, with exclusive control at the logical record level. Only one copy of CA-Datacom/DB is required to handle all regions. Recovery is centralized for all users.





## CHAPTER

## 3

## Defining SAS/ACCESS Descriptor Files

<i>Introduction to Defining SAS/ACCESS Descriptor Files</i>	15
<i>SAS/ACCESS Descriptor Files Essentials</i>	15
<i>Creating SAS/ACCESS Descriptor Files</i>	16
<i>Access Descriptor and View Descriptor Files</i>	16
<i>The ACCESS Procedure</i>	16
<i>Creating Access Descriptors and View Descriptors in One PROC Step</i>	16
<i>Updating Descriptor Files</i>	20
<i>Extracting CA-Datcom/DB Data with the ACCESS Procedure</i>	21

### Introduction to Defining SAS/ACCESS Descriptor Files

To use the SAS/ACCESS interface to CA-Datcom/DB, you must define special files that describe CA-Datcom/DB tables and data to SAS. These files are called SAS/ACCESS descriptor files. This section uses examples to illustrate creating and editing these files, as well as using the ACCESS procedure to extract CA-Datcom/DB data and place it in a SAS data file. (For complete reference information about the ACCESS procedure, see Chapter 6, “ACCESS Procedure Reference,” on page 59.)

The examples in this section are based on the CA-Datcom/DB table named CUSTOMERS. (See Appendix 3, “Data and Descriptors for the Examples,” on page 129 to review the data in this table.) The examples create an access descriptor file named MYLIB.CUSTS for that table. Then, the examples create two view descriptor files, VLIB.USACUST and VLIB.CUSTADD, based on the access descriptor.

### SAS/ACCESS Descriptor Files Essentials

SAS interacts with CA-Datcom/DB through an interface view engine that uses SAS/ACCESS descriptor files created with the ACCESS procedure. There are two types of descriptor files:

- access descriptor files (member type ACCESS)
- view descriptor files (member type VIEW)

An access descriptor contains information about the CA-Datcom/DB table that you want to use. The information includes the table name, the field names, and their data types. You use the access descriptor to create view descriptors. Think of an access descriptor as a master descriptor file for a single CA-Datcom/DB table, because it usually contains a complete description of that table.

A view descriptor defines a subset of the data described by an access descriptor. You choose this subset by selecting particular fields in the CA-Datcom/DB table, and you

can specify selection criteria that the data must meet. For example, you might want to select two fields, LAST-NAME and CITY-STATE, and specify that the value stored in field CITY-STATE must be **AUSTIN TX**. You can also specify a sequence order for the data. After you create your view descriptor, you can use it in a SAS program to read data directly from the CA-Datacom/DB table or to extract the data and place it in a SAS data file. Typically, for each access descriptor that you define, you have several view descriptors, each selecting different subsets of data.

---

## Creating SAS/ACCESS Descriptor Files

---

### Access Descriptor and View Descriptor Files

The examples in this section illustrate creating a permanent access descriptor named MYLIB.CUSTS and two view descriptors named VLIB.USACUST and VLIB.CUSTADD. Begin by using the SAS LIBNAME statement to associate librefs with the SAS data libraries in which you want to store the descriptors. (See the SAS documentation for your operating system for more details on the LIBNAME statement.)

You can have one library for access descriptors and a separate library for view descriptors, or you can put both access descriptors and view descriptors in the same library. Having separate libraries for access and view descriptors helps you maintain data security by enabling you to separately control who can read and update each type of descriptor.

In this document, the libref MYLIB is used for access descriptors and the libref VLIB is used for view descriptors.

---

### The ACCESS Procedure

You define descriptor files with the ACCESS procedure. You can define access descriptor files and view descriptor files in the same procedure execution or in separate executions. Within an execution, you can define multiple descriptors of the same or different types.

The following section shows how to define an access descriptor and multiple view descriptors in a single procedure execution. Examples of how to create the same descriptor files in separate PROC ACCESS executions are provided in Appendix 3, "Data and Descriptors for the Examples," on page 129.

When you use a separate PROC ACCESS execution to create a view descriptor, note that you must use the ACCDESC= option to specify an existing access descriptor from which the view descriptor will be created.

---

### Creating Access Descriptors and View Descriptors in One PROC Step

Perhaps the most common way to use the ACCESS procedure statements is to create an access descriptor and one or more view descriptors based on this access descriptor in a single PROC ACCESS execution. The following example shows how to do this. First an access descriptor is created (MYLIB.CUSTS). Then two view descriptors are created (VLIB.USACUST and VLIB.CUSTADD). Each statement is then explained in the order that it appears in the example program.

```
proc access dbms=Datacom;
  create mylib.custs.access;
```

```

user=demo;
table=customers;
assign = yes;
drop contact;
list all;
extend all;
rename customer = custnum telephone = phone
       streetaddress = street;
format firstorderdate = date7.;
informat firstorderdate = date7.;
content firstorderdate = yymmdd6.;
list all;

create vlib.usacust.view;
select customer state zipcode name
       firstorderdate;
list view;
extend view;

subset where customer eq 1#;
subset sort firstorderdate;
list view;

create vlib.custadd.view;
select state zipcode country name city;
list view;

list all;

run;

```

**proc access dbms=Datacom;**

invokes the ACCESS procedure for the SAS/ACCESS interface to CA-Datacom/DB.

**create mylib.custs.access;**

identifies the access descriptor, MYLIB.CUSTS, that you want to create. The MYLIB libref must be associated with a SAS library before you can specify this statement.

**user=demo;**

specifies a required CA-DATADictionary user ID. In this case, the user name is DEMO for the CA-Datacom/DB table CUSTOMERS. The name is the 32-character entity-occurrence name of a PERSON entity in CA-DATADictionary. The value entered is saved in the access descriptor and any view descriptor created from it. The user name and optional password (not used here) must have CA-DATADictionary retrieval authority on six entity-types: DATABASE, FILE, RECORD, ELEMENT, KEY, and FIELD.

**table=customers;**

indicates the name of the CA-Datacom/DB table that you want to use. The table name is required. The table name is a 32-character field that names an entity-occurrence of type RECORD in CA-DATADictionary. (For CA-Datacom/DB R8, the type is TABLE.) The combination of values in the TABLE statement and optional DATABASE and STATUS statements (not used here) must be unique.

**assign = yes;**

generates unique SAS column names based on the first eight non-blank characters of the CA-Datcom/DB field names. The column names and attributes can be changed in this access descriptor but not in any view descriptors created from this access descriptor.

Note that although the ASSIGN statement assigns names to the columns, it does not select them for inclusion in any view descriptors created from this access descriptor. You must select the fields in the view descriptor with the SELECT statement. Unless fields are dropped, they are automatically included in the access descriptor.

**drop contact;**

marks the CA-Datcom/DB field with the name CONTACT as non-display. The CONTACT field is a simple field; therefore, it is the only DBMS column that is dropped. When the DROP statement indicates a compound field, which can consist of multiple simple and compound fields, all DBMS columns associated with the compound field are marked as non-display, unless otherwise specified with the OCCURS statement. Compound fields are identified by the word \*GROUP\* in their description in the LIST statement output.

Columns that are dropped also do not appear in any view descriptors created from this access descriptor.

**list all;**

lists the access descriptor's item identifier numbers, the CA-Datcom/DB field names, the CA-Datcom/DB level numbers, the SAS column names, and the SAS formats. You can use the item identifier as a field identifier in statements that require you to use the DBMS column name. The list is written to the SAS log. Any columns that have been dropped from display (using the DROP statement) have \*NON-DISPLAY\* next to them.

**extend all;**

lists information about the SAS columns in the access descriptor, including the informat, the database content, and the number of times a field repeats. The list is written to the SAS log. When you are creating multiple descriptors, you can use the EXTEND statement before the next CREATE statement to list all the information about the descriptor you are creating.

**rename customer = custnum telephone = phone streetaddress = street;**

renames the default SAS column names associated with the CUSTOMER, TELEPHONE, and STREETADDRESS fields to CUSTNUM, PHONE, and STREET, respectively. Specify the CA-Datcom/DB field name or its positional equivalent from the LIST statement on the left side of the equal sign (=) and the new SAS name on the right. Because the ASSIGN=YES statement is specified, any view descriptors that are created from this access descriptor will automatically use the new names.

**format firstorderdate = date7.;**

changes the FIRSTORD SAS column from its default format to a new SAS format. The format specifies the way a value will be printed. In this case, it is a date format. Specify the CA-Datcom/DB field name or its positional equivalent from the LIST statement on the left side of the equal sign (=) and the new SAS format on the right. Because the ASSIGN=YES statement is specified, any view descriptors that are created from this access descriptor will automatically use the new format for the FIRSTORD column.

**informat firstorderdate = date7.;**

changes the FIRSTORD SAS column from its default informat to a new SAS informat. The informat specifies the way a value will be read. In this case, it is a

date informat. Specify the CA-Datcom/DB field name or its positional equivalent from the LIST statement on the left side of the equal sign (=) and the new informat on the right. Because the ASSIGN=YES statement is specified, any view descriptors that are created from this access descriptor will automatically use the new informat for the FIRSTORD column.

**content firstorderdate = yymmdd6.;**

specifies the SAS date format to use for the FIRSTORD SAS column. This format indicates the way date values are represented internally in the CA-Datcom/DB table (in this case, **yymmdd**). Specify the CA-Datcom/DB field name or its positional equivalent from the LIST statement on the left side of the equal sign (=) and the date format on the right. Because the ASSIGN=YES statement is specified, any view descriptors that are created from this access descriptor will automatically use this date format for the FIRSTORD column.

**list all;**

lists the item identifiers, the CA-Datcom/DB field names, the SAS column names, and other SAS information in the access descriptor so you can see the modifications before proceeding with the next CREATE statement.

**create vlib.usacust.view;**

writes the access descriptor to the library associated with MYLIB and identifies the view descriptor, VLIB.USACUST, that you want to create. The VLIB libref must be associated with a library before you can specify this statement.

**select customer state zipcode name firstorderdate;**

selects the CUSTOMER, STATE, ZIPCODE, NAME, and FIRSTORDERDATE fields for inclusion in the view descriptor. A SELECT statement is required to create the view, unless a RENAME, FORMAT, INFORMAT, or CONTENT statement is used.

**list view;**

lists the item identifiers, the DBMS column names, the SAS column names, and other SAS information associated with the CA-Datcom/DB fields selected for the view. The list is written to the SAS log.

**extend view;**

lists detail information about the SAS columns in the view, including the informat, the DB content, and the number of times a field repeats. The list is written to the SAS log.

**subset where customer eq 1#;**

specifies that you want to include only records with 1 as the first character in the CUSTOMER DBMS column.

**subset sort firstorderdate;**

specifies that you want to sort the records by the value of the FIRSTORDERDATE DBMS column.

**list view;**

lists the item identifiers, the DBMS column names, the SAS column names, and other SAS information associated with the view, to show the modifications.

**create vlib.custadd.view;**

writes view descriptor VLIB.USACUST to the library associated with VLIB and identifies a second view descriptor, VLIB.CUSTADD, that you want to create.

**select state zipcode country name city;**

selects the STATE, ZIPCODE, COUNTRY, NAME, and CITY fields for inclusion in the view descriptor.

**list view;**

lists the item identifiers, the DBMS column names, the SAS column names, and other SAS information associated with the CA-Datcom/DB fields selected for the view.

**list all;**

lists updated SAS information for the fields in the access descriptor. Fields that were dropped have \*NON-DISPLAY\* next to the SAS column description. Fields that were selected in the VLIB.CUSTADD view descriptor have \*SELECTED\* next to them. Fields that were selected in VLIB.USACUST will not show as selected in the access descriptor. Selection information, including status and any selection criteria, are reset in the access descriptor for each new view descriptor. The list is written to the SAS log.

**run;**

writes the view descriptor when the RUN statement is processed.

## Updating Descriptor Files

This section describes how to update existing descriptor files. You update access descriptor and view descriptor files with the UPDATE statement. You can edit the user and field information in the descriptor files.

When you update an access descriptor, the view descriptors that are based on this access descriptor are not updated automatically. You must re-create or update any view descriptors that you want to reflect the changes made to the access descriptor. That is, for some updates (such as dropping a field), the view descriptors are still valid, but they do not reflect the changes made in the access descriptor. In other situations (for example, if you edited the access descriptor to use a different userid or to add a password), the view descriptors would no longer be valid. A valid descriptor file can also be made useless by an update. For example, if an update to an access descriptor drops two of the four fields defined in a view descriptor, you might want to update or delete the view descriptor.

The following example updates access descriptor MYLIB.CUSTS to drop additional fields. The VLIB.USACUSTS and VLIB.CUSTADD view descriptors remain valid. However, you might want to update them to select new fields to replace those dropped as a result of the update.

```
proc access dbms=Datacom;
  update mylib.custs.access;
  drop 3 7;
  list all;
run;
```

The statements are described below.

**proc access dbms=Datacom;**

invokes the ACCESS procedure for the SAS/ACCESS interface to CA-Datcom/DB.

**update mylib.custs.access;**

identifies the access descriptor, MYLIB.CUSTS, that you want to update. The MYLIB libref must be associated with a SAS library before you can specify this statement.

**drop 3 7;**

marks the CA-Datcom/DB fields associated with position 3 (STATEZIP) and position 7 (TELEPHONE) as non-display. STATEZIP is a compound (\*GROUP\*)

field consisting of STATE and ZIPCODE. Dropping a group effectively drops the members of the group, so the STATE and ZIPCODE fields (which are selected in VLIB.USACUST and VLIB.CUSTADD) are marked as non-display as well.

**list all;**

lists updated SAS information for the fields in the access descriptor. Fields that were dropped have \*NON-DISPLAY\* next to the SAS column description. The list is written to the SAS log.

**run;**

writes the updated access descriptor when the RUN statement is processed.

Altering a CA-Datcom table that has descriptor files defined can also cause these files to be out of date or invalid. If you add a field to a table, an access descriptor is still valid. However, if you delete a field or change its characteristics and that field is used in a view descriptor, the view will fail when executed. For more information, see “How Changing the CA-DATADictionary Database Affects Descriptor Files” on page 109.

---

## Extracting CA-Datcom/DB Data with the ACCESS Procedure

Although you can access CA-Datcom/DB data directly in your SAS programs, it is sometimes better to extract the CA-Datcom/DB data and place it in a SAS data file. For example, if you are using the same CA-Datcom/DB data in several SAS jobs, it might be less resource-intensive to access extracted data in a SAS data file than to access a CA-Datcom/DB table repeatedly. (See “Performance Considerations” on page 41 for other circumstances in which extracting data is the more efficient method.)

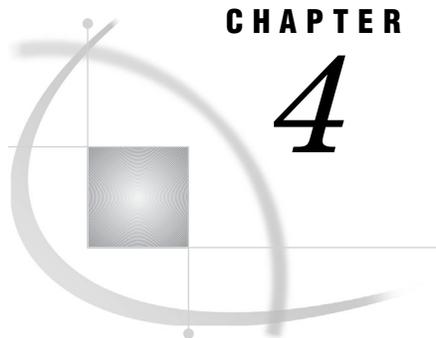
You can extract CA-Datcom/DB data by using PROC ACCESS statement options. You can also extract data using the DATA step. (See Chapter 4, “Using CA-Datcom/DB Data in SAS Programs,” on page 23 for examples using the SQL procedure to extract CA-Datcom/DB data and place it in a SAS data file.) Note that if you store view descriptors and SAS data files in the same SAS library, you must give them unique member names.

To extract data using the PROC ACCESS statement options, submit the following SAS code:

```
proc access viewdesc=vlib.usacust out=mydata.usaout;
run;
```

VLIB.USACUST is the two-level name that specifies the libref and member name for the view descriptor you want to use for extracting data, in this case, USACUST. Note that VLIB.USACUST must already exist. MYDATA.USAOUT is the two-level name specifying the libref and member name for the output SAS data file.





## CHAPTER

## 4

## Using CA-Datcom/DB Data in SAS Programs

---

<i>Introduction to Using CA-Datcom/DB Data in SAS Programs</i>	23
<i>Reviewing Columns for CA-Datcom/DB Data</i>	24
<i>Printing CA-Datcom/DB Data</i>	25
<i>Charting CA-Datcom/DB Data</i>	26
<i>Calculating Statistics for CA-Datcom/DB Data</i>	28
<i>Using FREQ, MEANS, and RANK Procedures</i>	28
<i>Using the FREQ Procedure</i>	28
<i>Using the MEANS Procedure</i>	28
<i>Using the RANK Procedure</i>	30
<i>Selecting and Combining CA-Datcom/DB Data</i>	30
<i>Using the WHERE Statement or the SQL Procedure</i>	30
<i>Selecting Data with the WHERE Statement</i>	31
<i>Combining Data with the SQL Procedure</i>	32
<i>Combining Data from Various Sources</i>	32
<i>Creating New Fields with the PROC SQL GROUP BY Clause</i>	37
<i>Updating a SAS Data File with CA-Datcom/DB Data</i>	38
<i>Using a DATA Step UPDATE Statement</i>	38
<i>Updating a Version 6 Data File</i>	38
<i>Updating a Version 8 and Later Data File</i>	40
<i>Performance Considerations</i>	41

---

### Introduction to Using CA-Datcom/DB Data in SAS Programs

An advantage of the SAS/ACCESS interface to CA-Datcom/DB is that it enables SAS to read and write CA-Datcom/DB data directly using SAS programs. This section presents examples using CA-Datcom/DB data that is described by view descriptors in SAS programs. For information about the views and sample data, see Appendix 3, “Data and Descriptors for the Examples,” on page 129.

Throughout the examples, the SAS terms *column* and *row* are used instead of comparable CA-Datcom/DB terms, because this section illustrates using SAS procedures and the DATA step. The examples include printing and charting data, using the SQL procedure to combine data from various sources, and updating Version 6 and Version 8 and later SAS data sets with data from CA-Datcom/DB. For more information about the SAS language and procedures used in the examples, refer to the documents listed at the end of each section.

At the end of this section, “Performance Considerations” on page 41 presents some techniques for using view descriptors efficiently in SAS programs.

## Reviewing Columns for CA-Datcom/DB Data

If you want to use CA-Datcom/DB data that is described by a view descriptor in your SAS program but cannot remember the SAS column names or formats and informats, you can use the CONTENTS or DATASETS procedure to display this information.

The following example uses the DATASETS procedure to give you information about the view descriptor VLIB.CUSPHON, which is based on the CA-Datcom/DB table CUSTOMERS.

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

The following output shows the information for this example. The data that is described by VLIB.CUSPHON is shown in Output 4.9.

**Output 4.1** Using the DATASETS Procedure with a View Descriptor

The SAS System							1
DATASETS PROCEDURE							
Data Set Name:	VLIB.CUSPHON	Observations:	22				
Member Type:	VIEW	Variables:	3				
Engine:	SASIODDB	Indexes:	0				
Created:	11:19 Friday, October 12, 1990	Observation Length:	80				
Last Modified:	12:03 Friday, October 12, 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 column labels using the DATASETS procedure. The labels are generated as the complete CA-Datcom/DB field name when the view descriptor is created, and they cannot be overridden.
- The **Created** date is when the access descriptor for this view descriptor was created.
- The **Last Modified** date is the last time the view descriptor was updated or created.
- The **Observations** number shown is the number of records in the CA-Datcom/DB table.

For more information about the DATASETS procedure, see the *SAS Language Reference: Dictionary* and the *Base SAS Procedures Guide*.

## Printing CA-Datcom/DB Data

Printing CA-Datcom/DB data that is described by a view descriptor is exactly like printing a SAS data file, as shown by the following example:

```
proc print data=vlib.empinfo;
  title2 'Brief Employee Information';
run;
```

VLIB.EMPINFO derives its data from the EMPLOYEES table. The following output shows the first page of output for this example.

**Output 4.2** Results of Printing CA-Datcom/DB Data

Brief Employee Information				1
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 row 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 rows described by the view descriptor VLIB.CUSORDR:

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

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

**Output 4.3** Results of Using the OBS= Option

First Five Data Records Described by VLIB.CUSORDR			1
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, but the FIRSTOBS= option does not improve performance significantly because each record must still be read and its position calculated.

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*.

---

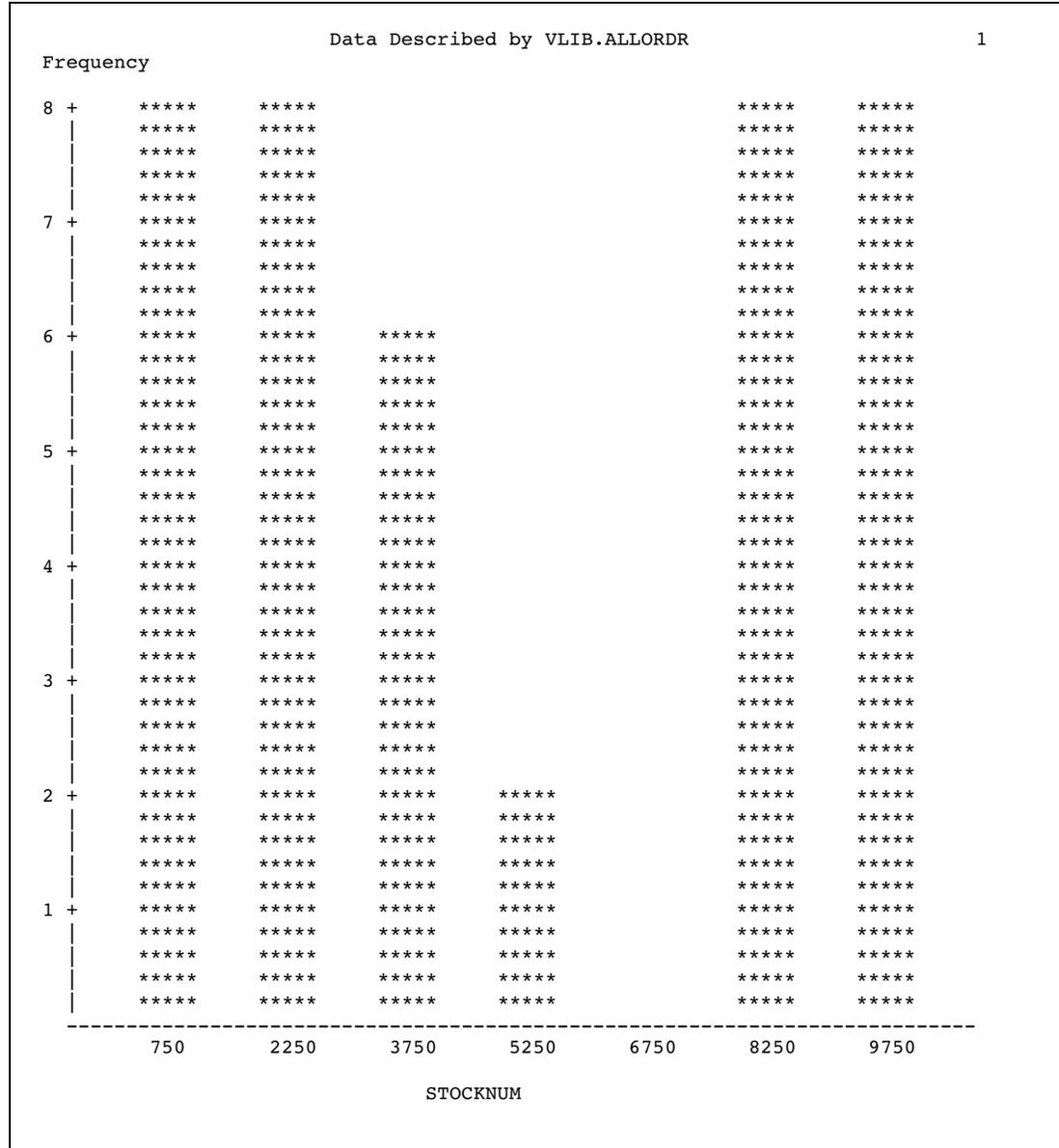
## Charting CA-Datcom/DB Data

CHART procedure programs work with CA-Datcom/DB data that is described by view descriptors just as they do with SAS data files. 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 table ORDER. The following output shows the information for this example. STOCKNUM represents each product. The number of orders for each product is represented by the height of the bar.

**Output 4.4** Results of Charting CA-Datcom/DB 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 CA-Datcom/DB data. See the *SAS/GRAPH Software: Reference, Volumes 1 and 2* for more information about the kinds of graphics you can produce with this SAS software product.

## Calculating Statistics for CA-Datcom/DB Data

### Using FREQ, MEANS, and RANK Procedures

You can use statistical procedures with CA-Datcom/DB data that is described by view descriptors just as you would with SAS data files. This section shows simple examples using the FREQ and MEANS procedures and a more advanced example using the RANK procedure.

### Using the FREQ Procedure

Suppose you want 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 percentage of invoices for each country appearing in the CA-Datcom/DB table 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 4.5** Results of Using the FREQ Procedure

Data Described by VLIB.INV					1
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	

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

### Using the MEANS Procedure

Suppose you want to determine some statistics for each of your USA customers. The view descriptor VLIB.USAORDR accesses records from the ORDER table that have a SHIPTO value beginning with a 1, indicating a USA customer.

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 rows (N) and the number of missing values (NMISS).

```
proc means data=vlib.usaordr mean sum n nmiss maxdec=0;
  by shipto;
  var length fabricch;
  title 'Data Described by VLIB.USAORDR';
run;
```

The BY statement causes the interface view engine to generate ordering criteria so that the data is sorted. The following output shows some of the information produced by this example.

**Output 4.6** Results of Using the MEANS Procedure

Data Described by VLIB.USAORDR						1
----- SHIPTO=14324742 -----						
Variable	Label	N	Nmiss	Mean	Sum	
LENGTH	LENGTH	4	0	1095	4380	
FABRICCH	FABRICCHARGES	2	2	1934460	3868920	
----- SHIPTO=14898029 -----						
Variable	Label	N	Nmiss	Mean	Sum	
LENGTH	LENGTH	2	0	2500	5000	
FABRICCH	FABRICCHARGES	2	0	1400825	2801650	
----- SHIPTO=15432147 -----						
Variable	Label	N	Nmiss	Mean	Sum	
LENGTH	LENGTH	4	0	725	2900	
FABRICCH	FABRICCHARGES	2	2	252149	504297	
----- SHIPTO=18543489 -----						
Variable	Label	N	Nmiss	Mean	Sum	
LENGTH	LENGTH	6	0	303	1820	
FABRICCH	FABRICCHARGES	4	2	11063836	44255344	
----- SHIPTO=19783482 -----						
Variable	Label	N	Nmiss	Mean	Sum	
LENGTH	LENGTH	4	0	450	1800	
FABRICCH	FABRICCHARGES	4	0	252149	1008594	
----- SHIPTO=19876078 -----						
Variable	Label	N	Nmiss	Mean	Sum	
LENGTH	LENGTH	2	0	690	1380	
FABRICCH	FABRICCHARGES	0	2	.	.	

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

---

## Using the RANK Procedure

You can use advanced statistics procedures with CA-Datcom/DB data that is described by a view descriptor. The following example uses the RANK procedure with data that is described by the view descriptor VLIB.EMPS 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. It assigns the column name DATERANK to the new field created by the procedure. (The VLIB.EMPS view descriptor includes a WHERE clause to select only the employees whose job code is 602.)

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

VLIB.EMPS is based on the CA-Datcom/DB table EMPLOYEES. The following output shows the result of this example.

**Output 4.7** Results of Using the RANK Procedure

OBS	EMPID	Order of Employee Birthdays			DATERANK	1
		JOBCODE	BIRTHDAT	LASTNAME		
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 CA-Datcom/DB Data

---

### Using the WHERE Statement or the SQL Procedure

Many SAS programs select and combine data from various sources. The method you use depends on the configuration of the data. The next examples show you how to select and combine data using two different methods. When choosing between these methods, consider the issues described in “Performance Considerations” on page 41.

## Selecting Data with the WHERE Statement

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

```
data notpaid(keep=invoice billedto amtbille billedon);
  set vlib.usainv vlib.forinv;
  where paidon is missing and amtbille >= 300000.00;
run;

proc print;
  title 'High Bills--Not Paid';
run;
```

In the SAS WHERE statement, be sure to use the SAS column names, not the CA-Datcom/DB field names. Both VLIB.USAINV and VLIB.FORINV are based on the CA-Datcom/DB table INVOICE. The following output shows the result of the new temporary data file, WORK.NOTPAID.

**Output 4.8** Results of Selecting Data with one WHERE Statement

OBS	INVOICEN	High Bills--Not Paid			1
		BILLEDTO	AMTBILLE	BILLEDON	
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 data set option works with SAS/ACCESS views just as it works with other SAS data sets. That is, the KEEP= option specifies that you want only the listed columns included in the new data file, NOTPAID, although you can use the other columns within the DATA step.

Notice that the WHERE statement includes two conditions to be met. First, it selects only rows that have a missing value for the field PAIDON. As you can see, it is important to know how the CA-Datcom/DB data is configured before you use this data in a SAS program. The field PAIDON contains values that translate to missing values in SAS. (Also, each of the two view descriptors has its own WHERE clause.)

Second, the WHERE statement requires that the amount in each bill be higher than a certain figure. Again, you should be familiar with the CA-Datcom/DB 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 WHERE statement than 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 a Boolean AND) to any WHERE clause defined in the view descriptor's selection criteria. The selection criteria are then passed to CA-Datcom/DB for processing. Processing CA-Datcom/DB data using a WHERE clause might reduce the number of records read from the database and therefore often improves performance.

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

---

## Combining Data with the SQL Procedure

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

---

## Combining Data from Various Sources

The SQL procedure provides a way to select and combine data from one or more database products. For example, suppose you have view descriptors VLIB.CUSPHON and VLIB.CUSORDR based on the CA-Datcom/DB tables CUSTOMERS and ORDER, respectively, and a SAS data file, MYDATA.OUTOFSTK, which contains product names and numbers that are out of stock. You can use the SQL procedure to join all these sources of data to form a single output file. A WHERE statement or a subsetting IF statement would not be appropriate in this case because you want to compare column values from several sources rather than simply merge or concatenate the data.

```
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 output shows the results of the PRINT procedure performed on the data that is described by the VLIB.CUSPHON and VLIB.CUSORDR view descriptors and on the MYDATA.OUTOFSTK SAS data file.

**Output 4.9** Data that is Described by the View Descriptor VLIB.CUSPHON

Data Described by VLIB.CUSPHON			1
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	DURHAM SCIENTIFIC SUPPLY COMPANY		
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 4.10** Data that is Described by the View Descriptor VLIB.CUSORDR

Data Described by VLIB.CUSORDR			1
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 4.11** Data in the SAS Data File MYDATA.OUTOFSTK

SAS Data File MYDATA.OUTOFSTK			1
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 view, `SQL.BADORDRS*`. This view retrieves customer and product information so that the sales department can notify customers of products that are no longer available.

```
proc sql;
create view sql.badorders 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.BADORDRS';
select * from sql.badorders;
```

The `CREATE VIEW` statement incorporates a `WHERE` clause as part of the `SELECT` statement, but it is not the same as the SAS `WHERE` statement illustrated earlier in this section. The last `SELECT` statement retrieves and displays the PROC SQL view, `SQL.BADORDRS`. To select all fields from the view, an asterisk (\*) is used in place of field names. The fields are displayed in the same order as they were specified in the first `SELECT` clause.

The following output shows the data that is described by the `SQL.BADORDRS` view. Note that the SQL procedure uses the DBMS labels in the output by default.

---

\* You might want to store your PROC SQL views in a SAS library other than the one storing your view descriptors, because they both have member type view.

## Output 4.12 Results of Combining DA-Datcom/DB Data

Data Described by SQL.BADORDRS			1
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.BADORDRS 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 SET statement and the special variable FIRST.PRODUCT. This variable identifies the first row in a particular BY group. You need a customer's name associated only once to notify that customer 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.badorders;
  by custnum product;
  if first.product;
run;

proc print;
  title 'MYDATA.BADNEWS Data File';
run;

```

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

**Output 4.13** Results of Subsetting Data with the FIRST Variable

MYDATA.BADNEWS Data File			1
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 the special variable FIRST, see “BY Statement” in the *SAS Language Reference: Dictionary*.

## Creating New Fields with the PROC SQL GROUP BY Clause

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

This example uses the SQL procedure to retrieve and manipulate data from the view descriptor VLIB.ALLEMP, which is based on the CA-Datcom/DB table EMPLOYEES. 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 columns displayed matches the order of the columns specified in the SELECT list of the query. The following output shows the query’s result.

**Output 4.14** Results of Creating New Fields with the SQL Procedure

Average Salary Per Department		1
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, refer to the *Base SAS Procedures Guide*.

---

## Updating a SAS Data File with CA-Datcom/DB Data

---

### Using a DATA Step UPDATE Statement

You can update a SAS data file with CA-Datcom/DB data that is described by a view descriptor the same way you update a SAS data file with data from another data file: by using a DATA step UPDATE statement. In this section, the term *transaction data* refers to the new data that is to be added to the original file. Because the SAS/ACCESS interface to CA-Datcom/DB uses the Version 6 compatibility engine, the transaction data is from a Version 6 source. The original file can be a Version 6 data file or a Version 8 and later data file.

---

### Updating a Version 6 Data File

You can update a Version 6 SAS data file with CA-Datcom/DB data the same way you did in Version 6 of SAS. 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 described by VLIB.EMPS, a view descriptor based on the CA-Datcom/DB table EMPLOYEES. To perform the update, enter the following SAS code:

```
proc sort data=lib6.birthday;
  by lastname;
run;
proc print data=lib6.birthday;
  format birthdat date7.;
  title 'LIB6.BIRTHDAY Data File';
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 updated SAS data file, MYDATA.NEWBDAY, is a Version 6 data file. It is stored in the Version 6 SAS 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 the interface view engine to automatically generate a SORT clause for the column LASTNAME. Thus, the SORT clause causes the CA-Datcom/DB data to be presented to SAS in a sorted order so it can be used to update the MYDATA.NEWBDAY data file. The data file LIB6.BIRTHDAY had to be sorted (by the SAS SORT procedure) before the update, because the UPDATE statement expects the data to be sorted by the BY column.

The following output shows the results of the PRINT procedure on the original data file, the transaction data, and the updated data file.

**Output 4.15** Data File to Be Updated, LIB6.BIRTHDAY

LIB6.BIRTHDAY Data File				1
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 4.16** Data that is Described by VLIB.EMPS

Data Described by VLIB.EMPS					1
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 4.17** Updated Data File, MYDATA.NEWBDAY

MYDATA.NEWBDAY Data File					1
OBS	EMPID	BIRTHDAT	LASTNAME	JOBCODE	
1	456910	24SEP53	ARDIS	602	
2	237642	13MAR54	BATTERSBY	602	
3	129540	31JUL60	CHOU LAI	.	
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	SHIEKELES LAN	.	
17	234967	21DEC67	SMITH	602	
18	254896	06APR49	TAYLOR-HUNYADI	.	
19	212916	29MAY28	WACHBERGER	602	
20	119012	05JAN46	WOLF-PROVENZA	602	

## Updating a Version 8 and Later Data File

Versions 6, 8, and later of SAS support different naming conventions; therefore, there could be character-length discrepancies between the columns in the original data file and the transaction data. You have two choices when updating a Version 8 and later data file:

- Let the compatibility engine truncate names exceeding 8 characters. The truncated names will be added to the updated data file as new columns.
- Rename the columns in the Version 8 and later data file to match the columns in the descriptor file.

The following example resolves character-length discrepancies by using the RENAME= DATA step option with the UPDATE statement. A Version 8 data file, LIB8.BIRTHDAYS, is updated with data described by VLIB.EMPS.

```
proc sort data=lib8.birthdays;
  by last_name;
run;

proc print data=lib8.birthdays;
  format birthdate date7.;
  title 'LIB8.BIRTHDAYS Data File';
run;

data newdata.v8_birthdays;
  update lib8.birthday
  (rename= (last_name=lastname
           first_name=firstnme
           birthdate=birthdat)) vlib.emps;
```

```

    by lastname firstnme;
run;

proc print data=newdata.v8_birthdays;
    title 'NEWDATA.V8_BIRTHDAYS Data File';
run;

```

In this example, the up-dated data file NEWDATA.V8\_BIRTHDAYS is a Version 8 data file that is stored in a Version 8 library associated with the libref NEWDATA. Version 8 and later supports member and column names of up to 32 characters. However, the RENAME= DATA step option is used with the UPDATE statement to change the longer column names in LIB8.BIRTHDAYS to match the 8-character column names in VLIB.EMPS. The columns are renamed *before* the updated data file is created.

The following output shows the results of the PRINT procedure on the original data file. The updated file looks like Output 4.17.

**Output 4.18** Data File to be Updated, LIB8.BIRTHDAYS

LIB8.BIRTHDAYS Data File				1
OBS	EMPLOYEE_ID	BIRTHDATE	LAST_NAME	
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	

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

You cannot update a CA-Datcom/DB table directly using the DATA step, but you can update a CA-Datcom/DB table using SAS/AF applications and the following procedures: APPEND, FSEDIT, FSVIEW, and SQL. See Chapter 5, “Browsing and Updating CA-Datcom/DB Data,” on page 43 for more information about updating CA-Datcom/DB data.

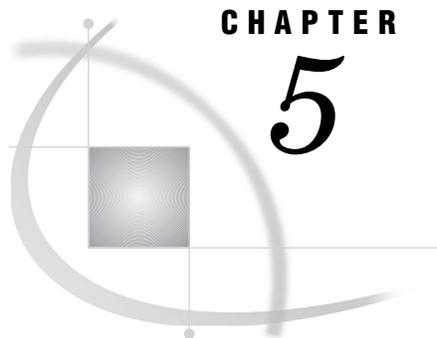
---

## Performance Considerations

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

- It is sometimes better to extract CA-Datcom/DB data and place it in a SAS data file than to read it directly. Here are some circumstances when you should probably extract:
  - If you plan to use the same CA-Datcom/DB data in several procedures in the same session, you might improve performance by extracting the CA-Datcom/DB 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 when they read CA-Datcom/DB data directly.

- If you plan to read large amounts of data from a CA-Datcom/DB table and the data is being shared by several users (multi-user environment), your direct reading of the data could adversely affect all users' response times.
  - If you are the creator of a table, and you 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 more efficient to run the SORT procedure on the view descriptor, using the OUT= option, than to request the same sort repeatedly (with a SORT clause) on the CA-Datcom/DB 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, or with a SORT clause included in the view descriptor. When you use a SAS BY statement with a view descriptor, it is most efficient to use a BY column that is associated with an indexed CA-Datcom/DB field. Also, if you do not need a certain order, blank out the Default Key. Otherwise, you might cause an unnecessary sort.
- If you use a Default Key, the interface view engine will use an index read instead of a sort if it can. Index reads are faster, but not always possible. For example, an index read is not possible if you specify multiple sort keys, multiple WHERE clause conditions, or a WHERE clause condition with a column that is not a key.
- When you are writing a SAS program and referencing a view descriptor, it is more efficient to use a WHERE statement in the program than it is to use a subsetting IF statement. The interface view engine passes the WHERE statement as CA-Datcom/DB selection criteria to the view descriptor, connecting it (with the AND operator) to any WHERE clause included in the view descriptor. Applying a WHERE clause to the CA-Datcom/DB data might reduce the number of records processed, which often improves performance.
- You can provide your own URT with options that are fine-tuned for your applications.
- Refer to "Creating and Using View Descriptors Efficiently" on page 96 for more details on creating efficient view descriptors.



## CHAPTER

## 5

## Browsing and Updating CA-Datcom/DB Data

<i>Introduction to Browsing and Updating CA-Datcom/DB Data</i>	<b>43</b>
<i>Browsing and Updating CA-Datcom/DB Data with the SAS/FSP Procedures</i>	<b>44</b>
<i>Using the FSBROWSE, FSEDIT, and FSVIEW Procedures</i>	<b>44</b>
<i>Browsing Data with PROC FSBROWSE</i>	<b>44</b>
<i>Updating Data with PROC FSEDIT</i>	<b>45</b>
<i>Browsing Data with PROC FSVIEW</i>	<b>45</b>
<i>Updating Data with PROC FSVIEW</i>	<b>46</b>
<i>Specifying a WHERE Clause While Browsing or Updating Data</i>	<b>47</b>
<i>Inserting and Deleting Data Records with the SAS/FSP Procedures</i>	<b>48</b>
<i>Browsing and Updating CA-Datcom/DB Data with the SQL Procedure</i>	<b>50</b>
<i>Using the SQL Procedure</i>	<b>50</b>
<i>Browsing Data with the SELECT Statement</i>	<b>50</b>
<i>Updating Data with the UPDATE Statement</i>	<b>52</b>
<i>Adding and Removing Data with the INSERT and DELETE Statements</i>	<b>53</b>
<i>Appending CA-Datcom/DB Data with the APPEND Procedure</i>	<b>53</b>

### Introduction to Browsing and Updating CA-Datcom/DB Data

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

Most of the examples in this section use the view descriptor VLIB.USACUST that you created in Chapter 3, “Defining SAS/ACCESS Descriptor Files,” on page 15. See Appendix 3, “Data and Descriptors for the Examples,” on page 129 for definitions of the view descriptors referenced in this section. This appendix also contains the CA-Datcom/DB tables and SAS data files used in this document.

Refer to Chapter 2, “CA-Datcom/DB Essentials,” on page 7 and Appendix 1, “Information for the Database Administrator,” on page 103 for more information about retrieval processing, update processing, and locks.

---

## Browsing and Updating CA-Datcom/DB Data with the SAS/FSP Procedures

---

### Using the FSBROWSE, FSEDIT, and FSVIEW Procedures

If your site has SAS/FSP software as well as SAS/ACCESS software, you can browse and update CA-Datcom/DB data that is described by a view descriptor from within a SAS program.

You have a choice of three SAS/FSP procedures: FSBROWSE, FSEDIT, and FSVIEW. The FSBROWSE and FSEDIT procedures show you one data record at a time, while the FSVIEW procedure displays multiple records in a tabular format similar to the PRINT procedure. PROC FSVIEW enables you to browse or update CA-Datcom/DB data, depending on which option you choose. You cannot use the FSBROWSE, FSEDIT, or FSVIEW procedures on an access descriptor.

To scroll through the data, use the FORWARD and BACKWARD commands. To end your browse or edit session, issue the END command.

---

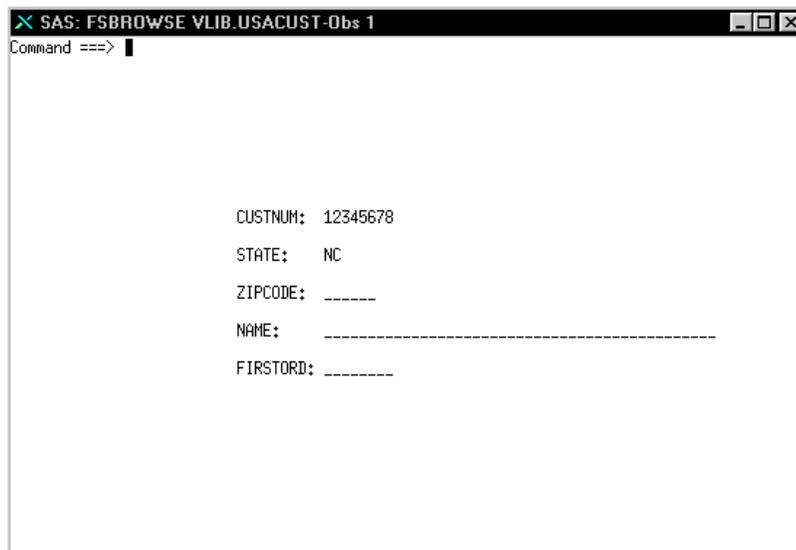
### Browsing Data with PROC FSBROWSE

The FSBROWSE procedure enables you to look at CA-Datcom/DB data that is described by a view descriptor but not to change it. For example, the following SAS statements enable you to look at one record at a time:

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

The FSBROWSE procedure retrieves one record at a time from a CA-Datcom/DB table. The following graphic shows the first record of the USA customers' data described by the VLIB.USACUST view descriptor. To browse each record, use the FORWARD and BACKWARD commands.

Display 5.1 FSBROWSE Window



---

## Updating Data with PROC FSEDIT

The FSEDIT procedure enables you to update CA-Datcom/DB data that is described by a view descriptor. For example, in Display 5.1 on page 44 the ZIPCODE, NAME, and FIRSTORD values are missing in the first record. You can add values to these fields with the FSEDIT procedure.

To use PROC FSEDIT, submit the following SAS statements:

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

Display 5.2 FSEDIT Window



The FSEDIT procedure also retrieves one record at a time. To edit a record, scroll to it, and type in the new data after the appropriate label. For example, enter the information about the **DURHAM SCIENTIFIC SUPPLY COMPANY**, as shown in Display 5.2 on page 45. To end your editing session, issue the END command.

*Note:* The data is presented in order by the Default Key value (usually the Native Key) unless the view descriptor contains a SORT clause.  $\Delta$

---

## Browsing Data with PROC FSVIEW

To browse CA-Datcom/DB data, submit the PROC FSVIEW statement as follows:

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

Display 5.3 FSVIEW Window

Obs	CUSTNUM	STATE	ZIPCODE	NAME	FIRSTORD
1	12345678	NC	27702	DURHAM SCIENTIFIC SUPPLY COMPANY	02JAN99
2	14324742	CA	95123	SANTA CLARA VALLEY TECHNOLOGY SPECIALISTS	05FEB85
3	14569877	NC	27514	PRECISION PRODUCTS	15AUG83
4	14898029	MD	20850	UNIVERSITY BIOMEDICAL MATERIALS	12NOV76
5	15432147	MI	49001	GREAT LAKES LABORATORY EQUIPMENT MANUFACTURER	28APR86
6	18543489	TX	78701	LONE STAR STATE RESEARCH SUPPLIERS	10SEP79
7	19783482	VA	22090	TWENTY-FIRST CENTURY MATERIALS	18JUL68
8	19876078	CA	93274	SAN JOAQUIN SCIENTIFIC AND INDUSTRIAL SUPPLY	11MAY79

PROC FSVIEW displays the data in a listing format instead of one observation at a time, as shown in the graphic above. Browse mode is the default for the FSVIEW procedure. Notice that a (B) for browse follows the view descriptor's name and that the values in the first record reflect the changes made using the FSEEDIT procedure in the previous example.

To see the rest of the table's data, scroll the display on the monitor to the right several times by issuing the RIGHT command on the command line or by using the function key assigned to this command.

*Note:* The data is presented in order by the Default Key value (usually the Native Key) unless the view descriptor contains a SORT clause. If the view descriptor contains a WHERE clause but no SORT clause, the order is unpredictable.  $\Delta$

---

## Updating Data with PROC FSVIEW

To edit CA-Datcom/DB data with PROC FSVIEW, submit the PROC FSVIEW statement as follows:

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

The word "EDIT" can be used instead of MODIFY. The display will be the same as Display 5.3 on page 46 except that an (E) for edit will be displayed in the window title.

*Note:* Any update in the FSVIEW window is final. The CANCEL command in the FSVIEW window does *not* cancel your changes, whether you have scrolled.  $\Delta$

---

## Specifying a WHERE Clause While Browsing or Updating Data

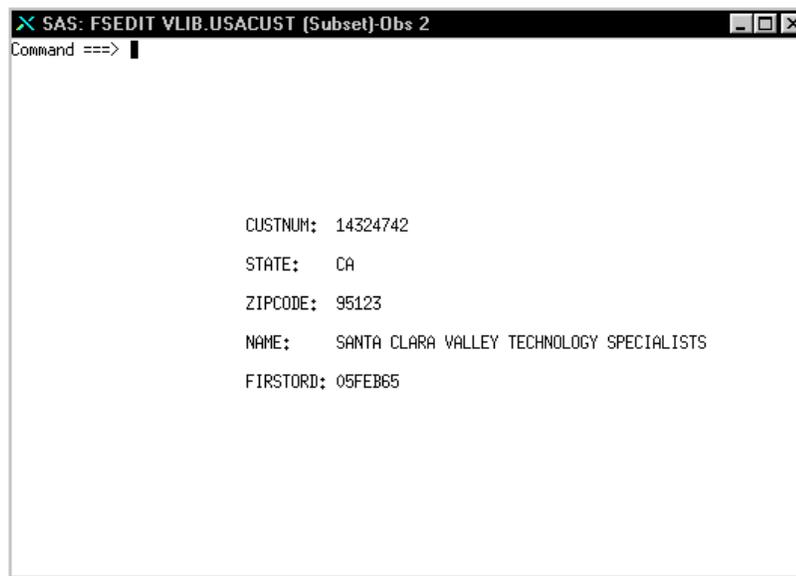
You can specify a WHERE statement to subset CA-Datcom/DB data when you invoke the SAS/FSP procedures. You can also use a WHERE command to do the same thing after you have invoked one of the SAS/FSP procedures.

In the following example, a WHERE statement is used to retrieve only customers from California:

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

The following graphic shows the FSEDIT window after the statements have been submitted.

**Display 5.4** FSEDIT Window After SAS WHERE Statement



Only two records with a STATE value of CA are retrieved for editing. Note that the word (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 record by typing over the information you want to modify. Issue the END command to end your editing session. If you want to cancel changes to a record, you can issue the CANCEL command before you scroll. Once you scroll though, the change is committed.

You can also use a SAS WHERE command to display a subset of your data. A WHERE command is a SAS WHERE expression that is entered on the command line. The following graphic shows how the FSEDIT window appears when the subset is generated within the procedure with the following WHERE command:

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

**Display 5.5** FSEDIT Window After SAS WHERE Command

Only the two records with a `STATE` value of `CA` are retrieved for editing. **where** 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 record, as described earlier.

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

---

## Inserting and Deleting Data Records with the SAS/FSP Procedures

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

You can use the `FSEDIT` and `FSVIEW` procedures to insert records into a CA-Datcom/DB table on which a view descriptor is based. Insertion of new records depends on the attributes assigned to the Master Key and whether the Master Key is included in your view descriptor. For example, if the `DUPE-MASTER-KEY` attribute is set to `N` (no), values for the Master Key cannot be duplicated. You will receive an error message if you try to insert a record having a Master Key value that duplicates an existing value. Therefore, be sure to define your view descriptors carefully if you intend to use them to insert records into a CA-Datcom/DB table.

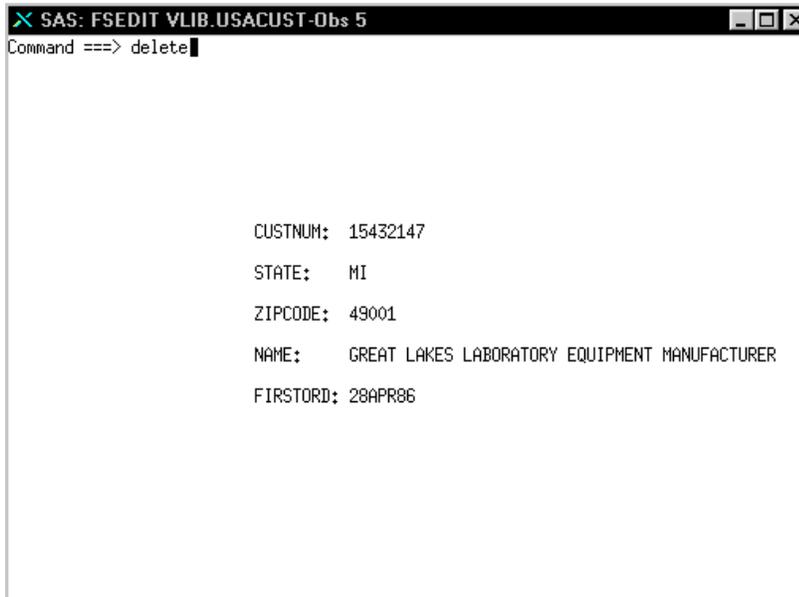
Refer to Appendix 1, “Information for the Database Administrator,” on page 103 for details about inserting data. Refer to *SAS/FSP Procedures Guide* for information about how to use insertion commands such as `ADD` and `DUP` in the `FSEDIT` procedure and `AUTOADD` and `DUP` in the `FSVIEW` procedure. However, note that with the SAS/ACCESS interface to CA-Datcom/DB, a duplicated record is inserted immediately after the original record rather than at the end of the CA-Datcom/DB table.

When you use the `DELETE` command with a view descriptor that describes CA-Datcom/DB data, the current record is removed permanently from the CA-Datcom/DB table. Also, the `DELETE` command works differently in the `FSVIEW`

procedure than it does in the FSEDIT procedure. Refer to *SAS/FSP Procedures Guide* for more information about this command.

The following example illustrates using the DELETE command in the FSEDIT procedure. Scroll forward to the record to be deleted and enter the DELETE command on the command line, as shown in the following graphic.

**Display 5.6** Deleting a CA-Datcom/DB Record



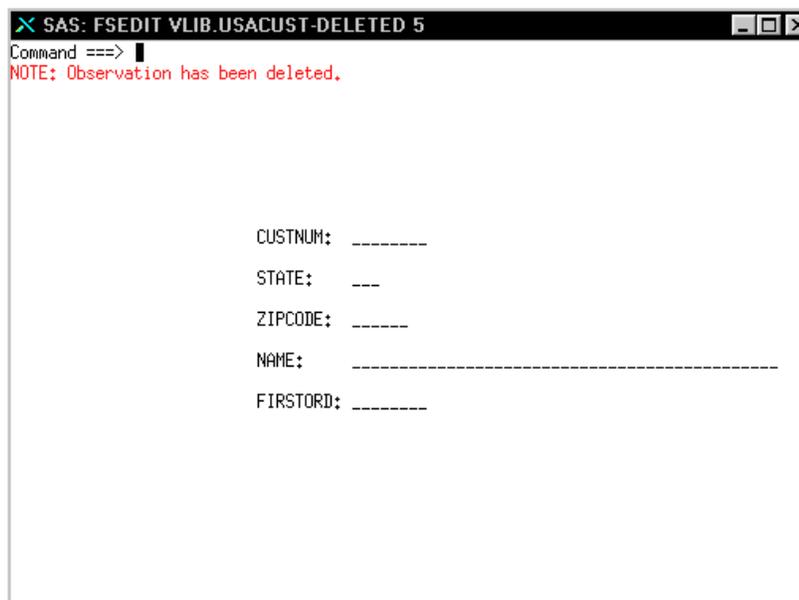
```

X SAS: FSEDIT VLIB.USACUST-Obs 5
Command ==> delete

                CUSTNUM: 15432147
                STATE:  MI
                ZIPCODE: 49001
                NAME:   GREAT LAKES LABORATORY EQUIPMENT MANUFACTURER
                FIRSTORD: 28APR86
  
```

The DELETE command deletes the record and displays a message to that affect, as shown in the following graphic.

**Display 5.7** The Deleted Row



```

X SAS: FSEDIT VLIB.USACUST-DELETED 5
Command ==> 
NOTE: Observation has been deleted.

                CUSTNUM:  -----
                STATE:  ---
                ZIPCODE:  -----
                NAME:   -----
                FIRSTORD:  -----
  
```

For more information about using the SAS/FSP procedures, see the *SAS/FSP Procedures Guide*.

---

## Browsing and Updating CA-Datcom/DB Data with the SQL Procedure

---

### Using the SQL Procedure

The SAS SQL procedure also enables you to retrieve and update CA-Datcom/DB data. You can retrieve and browse the 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 INSERT, DELETE, and UPDATE statements. Here is a summary of these PROC SQL statements:

DELETE	deletes records from a CA-Datcom/DB table.
INSERT	inserts records into a CA-Datcom/DB table.
SELECT	retrieves and displays data from CA-Datcom/DB tables. A SELECT statement is usually referred to as a query, because it queries the tables for information.
UPDATE	updates records in a CA-Datcom/DB table.

When using the SQL procedure in interactive line mode, 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. You can use the QUIT statement if you want to exit the SQL procedure.

#### **CAUTION:**

**When you use the SQL procedure for update processing (DELETE, INSERT, and UPDATE statements), you must set the SQL procedure option UNDO\_POLICY.** The SQL procedure supports backouts of group updates for those databases that support member-level locking. CA-Datcom/DB software does not support member-level locks. The UNDO\_POLICY option enables updates to be processed without backouts. For the CA-Datcom/DB interface, you set the value of the option to NONE. For example:

```
proc sql undo_policy=none;
  update vlib.usacust
  set zipcode=27702
  where custnum='12345678';
```

If the update is processed successfully, it is applied to the database table and a warning message is issued. The message signifies that if multiple records were updated by the command and a failure occurred some time after the first record was successfully processed, then there is no way for PROC SQL to avoid a partial update.

Partial updating means that some records are updated and some are not. It does not mean that some fields in the same record are updated while other fields are not.  $\Delta$

---

### Browsing Data with the SELECT Statement

You can use the SELECT statement to browse CA-Datcom/DB data that is described by a view descriptor. The query in the following example retrieves and displays all the fields and records in the CUSTOMERS table that are described by the VLIB.USACUST

view descriptor. The UNDO\_POLICY option is included to disable member-level locking and to enable updates later in the PROC SQL execution. You can exclude the UNDO\_POLICY option if you do not plan to perform updates. 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 table has not been updated by the earlier SAS/FSP examples.  $\Delta$

```
options linesize=120;

proc sql undo_policy=none;
  title 'CA---Datcom/DB Data Output from a SELECT Statement';
select custnum, state label='STATE', zipcode label='ZIPCODE',
       name, firstord
  from vlib.usacust;
```

The following output shows the query's results. Notice that the SQL procedure displays the CA-Datcom/DB field names, not the corresponding SAS column names.

**Output 5.1** Results of a PROC SQL Query

CA-Datcom/DB Data Output from a SELECT Statement				
CUSTOMER	STATE	ZIPCODE	NAME	FIRSTORDERDATE
12345678	NC	.		.
14324742	CA	95123	SANTA CLARA VALLEY TECHNOLOGY SPECIALISTS	05FEB65
19783482	VA	22090	TWENTY-FIRST CENTURY MATERIALS	18JUL68
14898029	MD	20850	UNIVERSITY BIOMEDICAL MATERIALS	12NOV76
19876078	CA	93274	SAN JOAQUIN SCIENTIFIC AND INDUSTRIAL SUPPLY, INC.	11MAY79
18543489	TX	78701	LONE STAR STATE RESEARCH SUPPLIERS	10SEP79
14569877	NC	27514	PRECISION PRODUCTS	15AUG83
15432147	MI	49001	GREAT LAKES LABORATORY EQUIPMENT MANUFACTURERS	28APR86

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

```
title 'CA---Datcom/DB Data Output Subset by a WHERE Clause';
select custnum, state label='STATE', zipcode label='ZIPCODE',
       name, firstord
  from vlib.usacust
 where state='NC';
```

Notice that the PROC SQL statement is not repeated in this query. You do not need to repeat the PROC statement unless you use another SAS procedure, the DATA step, or a QUIT statement between PROC SQL statements. The following output displays the two companies from North Carolina described by VLIB.USACUST.

**Output 5.2** Results of PROC SQL Query Subset by a WHERE Clause

CA-Datcom/DB Data Output Subset by a WHERE Clause					
CUSTOMER	STATE	ZIPCODE	NAME	FIRSTORDERDATE	
12345678	NC	.			.
14569877	NC	27514	PRECISION PRODUCTS		15AUG83

## Updating Data with the UPDATE Statement

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

The following UPDATE statements update the values described by the first record of VLIB.USACUST. The SELECT statement then displays the view's output. The ORDER BY clause in the SELECT statement causes the data to be presented in ascending order by the CUSTNUM field. The UNDO\_POLICY option is omitted since it was specified in the original SQL request.

```

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';
  title 'Updated VLIB.USACUST View Descriptor';
select custnum, state label='STATE', zipcode label='ZIPCODE', name,
       firstord from vlib.usacust
       order by custnum;

```

The following output displays the query's results.

**Output 5.3** Results of Updating Data with an UPDATE Statement

Updated VLIB.USACUST View Descriptor					
CUSTOMER	STATE	ZIPCODE	NAME	FIRSTORDERDATE	
12345678	NC	27702	DURHAM SCIENTIFIC SUPPLY COMPANY		02JAN88
14324742	CA	95123	SANTA CLARA VALLEY TECHNOLOGY SPECIALISTS		05FEB65
14569877	NC	27514	PRECISION PRODUCTS		15AUG83
14898029	MD	20850	UNIVERSITY BIOMEDICAL MATERIALS		12NOV76
15432147	MI	49001	GREAT LAKES LABORATORY EQUIPMENT MANUFACTURERS		28APR86
18543489	TX	78701	LONE STAR STATE RESEARCH SUPPLIERS		10SEP79
19783482	VA	22090	TWENTY-FIRST CENTURY MATERIALS		18JUL68
19876078	CA	93274	SAN JOAQUIN SCIENTIFIC AND INDUSTRIAL SUPPLY, INC.		11MAY79

## Adding and Removing Data with the INSERT and DELETE Statements

You can use the INSERT statement to add records to a CA-Datcom/DB table or the DELETE statement to remove records. In the following example, the record containing the CUSTNUM value 15432147 is deleted from the table CUSTOMERS. The SELECT statement then displays the VLIB.USACUST data, ordering them again by the CUSTNUM field. Again, the UNDO\_POLICY option was omitted because it was specified in the original SQL request and no intervening SAS procedure, DATA step, or QUIT statement occurred between SQL statements.

```
delete from vlib.usacust
  where custnum='15432147';
title 'Record Deleted from CA-Datcom/DB CUSTOMERS Table';
select custnum, state label='STATE', zipcode label='ZIPCODE',
       name, firstord
  from vlib.usacust
 order by custnum;
```

The following output displays the query's results.

**Output 5.4** Results of Removing Data with a DELETE Statement

Record Deleted from CA-Datcom/DB CUSTOMERS Table				
CUSTOMER	STATE	ZIPCODE	NAME	FIRSTORDERDATE
12345678	NC	27702	DURHAM SCIENTIFIC SUPPLY COMPANY	02JAN88
14324742	CA	95123	SANTA CLARA VALLEY TECHNOLOGY SPECIALISTS	05FEB65
14569877	NC	27514	PRECISION PRODUCTS	15AUG83
14898029	MD	20850	UNIVERSITY BIOMEDICAL MATERIALS	12NOV76
18543489	TX	78701	LONE STAR STATE RESEARCH SUPPLIERS	10SEP79
19783482	VA	22090	TWENTY-FIRST CENTURY MATERIALS	18JUL68
19876078	CA	93274	SAN JOAQUIN SCIENTIFIC AND INDUSTRIAL SUPPLY, INC.	11MAY79

### CAUTION:

**Always use the WHERE clause in a DELETE statement.** If you omit the WHERE clause from the DELETE statement, you will delete *all* the data in the CA-Datcom/DB table accessed by the view descriptor.  $\Delta$

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

## Appending CA-Datcom/DB Data with the APPEND Procedure

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.

The input file and base file do not have to match column for column. If they do not match, use the FORCE option in the APPEND procedure. This will include all columns in the base file. Values for columns that are not shared by the base and input files are set to missing.

In the following example, two personnel managers have kept separate employee records. One manager has kept records in the CA-Datcom/DB table EMPLOYEES, that is described by the view descriptor VLIB.DCMEMPS. The other manager has kept records in the SAS data file, MYDATA.SASEMPS. Due to a corporate reorganization, the two sources of data must be combined so that all employee data is stored in the CA-Datcom/DB table EMPLOYEES. The APPEND procedure can perform this task.

The data that is described by the view descriptor VLIB.DCMEMPS and the data in the SAS data file MYDATA.SASEMPS are printed with the following statements and displayed in Output 5.5 and Output 5.6.

```
proc print data=vlib.dcmemps;
    title 'Data Described by VLIB.DCMEMPS';
run;

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

**Output 5.5** Data Described by VLIB.DCMEMPS

OBS	EMPID	BIRTHDAT	Data Described by VLIB.DCMEMPS			1
			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	15JAN34	RODRIGUES	JUAN	M.	
22	677890	24APR65	NISHIMATSU-LYNCH	CAROL	ANNE	

**Output 5.6** Data in MYDATA.SASEMPS

Data in MYDATA.SASEMPS Data File						1
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	

Submitting the following APPEND procedure combines data from these two sources:

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

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

The following output displays the appended data that is described by the view descriptor VLIB.DCMEMPS. Notice that the data is inserted in the order of Native Key values.

**Output 5.7** Result of Appending Data

Appended Data						1
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	219223	13JUN47	HANSINGER	BENJAMIN	HAROLD	
10	234967	21DEC67	SMITH	GILBERT	IRVINE	
11	237642	13MAR54	BATTERSBY	R.	STEPHEN	
12	239185	28AUG59	DOS REMEDIOS	LEONARD	WESLEY	
13	245962	30AUG64	BEDORTHA	KATHY	MARTHA	
14	254896	06APR49	TAYLOR-HUNYADI	ITO	MISHIMA	
15	321783	03JUN35	GONZALES	GUILLERMO	RICARDO	
16	326745	21FEB52	RAWN	BEATRICE	MAY	
17	328140	02JUN51	MEDINA-SIDONIA	MARGARET	ROSE	
18	346917	15MAR50	SHIEKELESLAM	SHALA	Y.	
19	356134	25OCT60	DUNNETT	CHRISTINE	MARIE	
20	423286	31OCT64	MIFUNE	YUKIO	TOSHIRO	
21	456910	24SEP53	ARDIS	RICHARD	BINGHAM	
22	456921	12MAY62	KRAUSE	KARL-HEINZ	G.	
23	457232	15OCT63	LOVELL	WILLIAM	SINCLAIR	
24	459287	05JAN34	RODRIGUES	JUAN	M.	
25	677890	24APR65	NISHIMATSU-LYNCH	CAROL	ANNE	
26	765432	01MAR59	POWELL	FRANK	X.	

The APPEND procedure also accepts a WHERE= data set option or a WHERE statement to retrieve a subset of data. In the following example, a subset of

observations from MYDATA.SASEMPS is added to VLIB.DCEMPS. The results are displayed in Output 5.8.

```
proc append base=vlib.dcmemps data=mydata.sasemps
  (where=(lastname like 'B%' or lastname like 'H%'));
run;

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

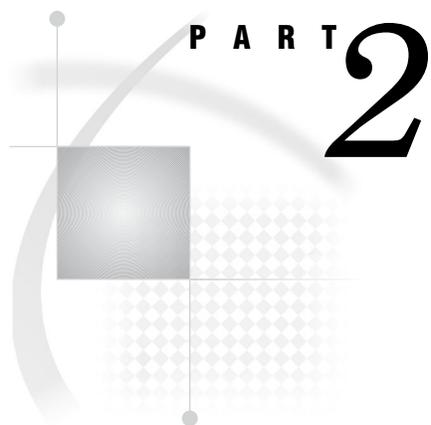
The following output displays the data when the observations appended to the BASE= data set are subset by the WHERE= data set option. In this case, the WHERE= data set option specifies that only the employees with last names beginning with B or H should be added to the BASE= data set.

**Output 5.8** Results of Appending Data with a WHERE= Data Set Option

OBS	EMPID	BIRTHDAT	Appended Data			1
			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	219223	13JUN46	HANSINGER	BENJAMIN	HAROLD	
10	234967	21DEC67	SMITH	GILBERT	IRVINE	
11	237642	13MAR54	BATTERSBY	R.	STEPHEN	
12	239185	28AUG59	DOS REMEDIOS	LEONARD	WESLEY	
13	245962	30AUG64	BEDORTHA	KATHY	MARTHA	
14	254896	06APR49	TAYLOR-HUNYADI	ITO	MISHIMA	
15	321783	03JUN35	GONZALES	GUILLERMO	RICARDO	
16	328140	02JUN51	MEDINA-SIDONIA	MARGARET	ROSE	
17	346917	15MAR50	SHIEKELESLAM	SHALA	Y.	
18	356134	25OCT60	DUNNETT	CHRISTINE	MARIE	
19	423286	31OCT64	MIFUNE	YUKIO	TOSHIRO	
20	456910	24SEP53	ARDIS	RICHARD	BINGHAM	
21	456921	12MAY62	KRAUSE	KARL-HEINZ	G.	
22	457232	15OCT63	LOVELL	WILLIAM	SINCLAIR	
23	459287	05JAN34	RODRIGUES	JUAN	M.	
24	677890	24APR65	NISHIMATSU-LYNCH	CAROL	ANNE	

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

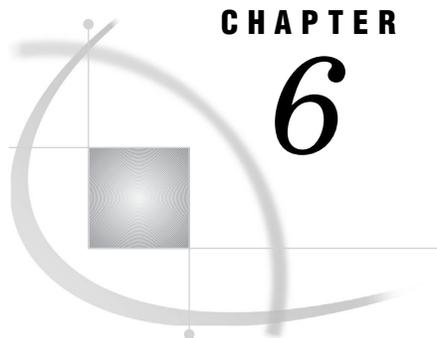
Note that when the FORCE option is used to append columns whose names do not match, any column names that are longer than 8 characters will be truncated at 8 characters.



## **SAS/ACCESS Interface to CA-Datcom/DB: Reference**

*Chapter 6* . . . . . **ACCESS Procedure Reference** 59





## CHAPTER

## 6

## ACCESS Procedure Reference

---

<i>Introduction to ACCESS Procedure Reference</i>	59
<i>ACCESS Procedure Syntax</i>	60
<i>Description</i>	61
<i>PROC ACCESS Statement Options</i>	61
<i>Options</i>	62
<i>SAS Passwords for SAS/ACCESS Descriptors</i>	63
<i>Overview of SAS Passwords</i>	63
<i>Assigning Passwords for SAS/ACCESS Descriptors</i>	63
<i>ACCESS Procedure Method for Assigning Passwords</i>	63
<i>DATASETS Procedure Method for Assigning Passwords</i>	64
<i>Invoking the ACCESS Procedure</i>	65
<i>Statements</i>	67
<i>WHERE Clause in a View Descriptor</i>	89
<i>View WHERE Clause Syntax</i>	89
<i>The Asterisk in View WHERE Clauses</i>	91
<i>View WHERE Clause Expressions</i>	91
<i>Specifying Values in View WHERE Clauses</i>	92
<i>Character Fields in View WHERE Clauses</i>	92
<i>Date Values in View WHERE Clauses</i>	92
<i>\$HEX. Format Fields in View WHERE Clauses</i>	92
<i>Values That Do Not Fit the Field Picture</i>	93
<i>Masking Values in View WHERE Clauses</i>	93
<i>Multi-Field Keys in View WHERE Clauses</i>	94
<i>Guidelines for View WHERE Clauses</i>	94
<i>SORT Clause in a View Descriptor</i>	95
<i>Overview of the SORT Clause</i>	95
<i>View SORT Clause Syntax</i>	95
<i>View SORT Clause Example</i>	96
<i>View SORT Clause Guidelines</i>	96
<i>Creating and Using View Descriptors Efficiently</i>	96
<i>ACCESS Procedure Data Conversions</i>	97

---

### Introduction to ACCESS Procedure Reference

The ACCESS procedure enables you to create and edit the descriptor files that are used by the SAS/ACCESS interface to CA-Datcom/DB. 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 use of the interface:

- “Creating and Using View Descriptors Efficiently” on page 96 presents several efficiency considerations for using the SAS/ACCESS interface to CA-Datcom/DB.
- “ACCESS Procedure Data Conversions” on page 97 summarizes how the SAS/ACCESS interface converts each type of CA-Datcom/DB data into its SAS column format and informat equivalents.

For examples of how to use PROC ACCESS, refer to Chapter 3, “Defining SAS/ACCESS Descriptor Files,” on page 15. If you need help with SAS data sets and data libraries, their naming conventions, or any terms used in the ACCESS procedure, refer to the *SAS Language Reference: Dictionary* and the *SAS Companion for z/OS*.

Remember that help is available from within the ACCESS procedure by issuing the HELP command on any command line.

---

## ACCESS Procedure Syntax

**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

**DATABASE** | **DB***<=>* *<">Datcom-database-name<">*;

**DBSTAT***<=>* *<">PROD<">* | *<">TEST<">* | *<">test-version<">*;

**PASSWORD** | **PASS** | **PW***<=>* *<">Datcom-password<">*;

**TABLE***<=>* *<">Datcom-table-name<">*;

**TBLSTAT***<=>* *<">PROD<">* | *<">TEST<">* | *<">test-version<">*;

**URT***<=>* *<">User-Requirements-Table-name<">*;

**USER***<=>* *<">authorized-Datcom-userid<">*;

### Editing Statements

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

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

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

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

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

**INFORMAT** | **INFMT** *<">column-identifier-1<">* *<=>* *SAS-format-name*  
*<...<">column-identifier-n<">* *<=>* *SAS-format-name*;

**KEY***<=>* *<">Datcom-short-name<">*;

**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<">>;
OCCURS <">column-identifier<">
  CONTENT occurrence-1 <=> SAS-format-name
  <...occurrence-n <=> SAS-format-name>;
  |
  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>;
  |
  RENAME <">occurrence-1<"> <=> SAS-name
  <...<">occurrence-n<"> <=> SAS-name>;
  |
  RESET occurrence-1 <TO> occurrence-n;
  |
  SELECT occurrence-1 <TO> occurrence-n;
RENAME <">column-identifier-1<"> <=> SAS-name
  <...<">column-identifier-n<"> <=> SAS-name>;
RESET ALL | <">column-identifier-1<"> <...<">column-identifier-n<">>;
SELECT ALL | <">column-identifier-1<"> <...<">column-identifier-n<">>;
SUBSET selection-criteria;
QUIT | EXIT;

```

---

## 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 can run in batch or interactive line 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 65 for information about which procedure statements to use for each task.

The PROC ACCESS statement takes the following options:

**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 SAS 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=Datacom**

specifies the database management system that you want the descriptor(s) to access. Specify DBMS=Datacom since you are using the SAS/ACCESS interface to CA-Datacom/DB.

**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 CA-Datacom/DB data. VIEWDESC= is used only with the OUT= option.

For example:

```
proc access dbms=Datacom viewdesc=vlib.invq4
    out=dlib.invq4;
run;
```

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

## Options

The ACCESS procedure statement takes the following options:

**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 SAS 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=Datacom**

specifies the database management system you want the descriptor(s) to access. Specify DBMS=Datacom since you are using the SAS/ACCESS interface to CA-Datacom/DB.

**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 CA-Datacom/DB data. VIEWDESC= is used only with the OUT= option.

For example:

```
proc access dbms=Datacom viewdesc=vlib.invq4
    out=dlib.invq4;
run;
```

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

---

## SAS Passwords for SAS/ACCESS Descriptors

---

### Overview of SAS Passwords

SAS enables you to control access to SAS data sets and access descriptors by associating one or more SAS passwords with them.

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

**Table 6.1** Password and Descriptor Interaction

	<b>READ=</b>	<b>WRITE=</b>	<b>ALTER=</b>
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

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 for SAS/ACCESS Descriptors

#### ACCESS Procedure Method for Assigning Passwords

You can assign a SAS password when you define a descriptor in the ACCESS procedure or after the descriptor file has been created by using PROC DATASETS.

Four password levels are available: READ=, WRITE=, ALTER=, and PW=. PW= assigns read, write, and alter privileges to a descriptor.

You can assign multiple levels of protection to a descriptor. 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 assign a password in the ACCESS procedure, specify the password level and password as a data set option in the CREATE statement. The following example creates and assigns passwords to an access descriptor and a view descriptor in the same procedure execution:

```
proc access dbms=Datacom;
  create work.emps.access (alter=rouge);
  table=employees;
  user=demo;

  create work.emp.view (alter=ego);
  select 1 2 3 4;
run;
```

Users will have to specify the ALTER password EGO to browse or edit the view descriptor and the ALTER password ROUGE to browse, edit, or define additional view descriptors from this access descriptor.

When creating a view descriptor from a password-protected access descriptor, specify the access descriptor password as a data set option after the ACCDESC= option. The following example specifies two data set options. The first specifies the access descriptor password and the second assigns a password to the view descriptor.

```
proc access dbms=Datacom ad=work.emps.access (alter=rouge);
  create work.emp2.view (alter=dumb);
  select 5 6 7 8;
run;
```

## DATASETS Procedure Method for Assigning Passwords

You assign a SAS password to an existing descriptor by using the DATASETS procedure. The DATASETS procedure MODIFY statement enables you to assign, change, and delete SAS passwords.

Here is the basic syntax for using PROC DATASETS to assign a password to an access descriptor, a view descriptor, or a SAS data file:

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

```
RUN;
```

In this syntax statement, the *password-level* argument can have one or more of the following values: READ=, WRITE=, ALTER=, or PW=. 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 MYLIB.EMPLOYEE.

```
proc datasets library=mylib memtype=access;
  modify employee (alter=money);
run;
```

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

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

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

In this case, users are prompted for the SAS password when they try to read or update the DBMS data, or try to browse or edit the view descriptor VLIB.CUSPHON itself. You need both levels to protect the data and descriptor. Assign a WRITE level of protection to prevent data updates.

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

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

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

---

## Invoking the ACCESS Procedure

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

- To create an access descriptor, use the following statements:

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

```
RUN;
```

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

```
PROC ACCESS DBMS=Datacom;
      CREATE libref.member-name.ACCESS;
          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, use the following statements:

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

```
RUN;
```

- To update an access descriptor, use the following statements:

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

```
RUN;
```

- To update a view descriptor, use the following statements:

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

```
RUN;
```

See for a listing of database description and editing statements. For information to help you code efficient descriptor files, see .

Note that when you update an access descriptor (for example, drop another field from the display), the view descriptors based on this access descriptor are not updated automatically. You must re-create or modify any view descriptors that you want to reflect the changes made to the access descriptor. Altering a DBMS table can invalidate both access descriptors and view descriptors.

**CAUTION:**

**Updating access descriptors does not automatically update view descriptors.** When you update an access descriptor (for example, drop another field from the display), the view descriptors based on this access descriptor are not updated automatically. You must re-create or modify any view descriptors that you want to reflect the changes made to the access descriptor. The view descriptors would still be valid, but they would no longer match the access descriptor. However, in some situations the view descriptors would no longer be valid (for example, if you re-create an access descriptor with the same name but base it on a different CA-Datacom/DB table).  $\Delta$

**CAUTION:**

**Altering CA-Datacom/DB tables can affect descriptor files.** Altering a CA-Datacom/DB table that has descriptor files defined on it might cause these descriptors to be out-of-date or invalid. For example, if you add a field to a table and an existing access descriptor is defined on that table, the access descriptor does not reflect the new field, but it remains valid. However, if you delete a field or delete a table on which the view descriptor is based, the view descriptor fails when executed. Therefore, you must change the descriptor files manually when changes to CA-DATADictionary invalidate them.

- 1 When you change CA-DATADictionary, you must re-create the access descriptor(s) with PROC ACCESS, using the same name(s).
- 2 Then you must edit each view descriptor with PROC ACCESS. You will get a message if the view descriptor differs from its access descriptor. Change the view descriptor as needed.

$\Delta$

The SAS/ACCESS interface view engine does a rudimentary validation of a view descriptor upon opening it. For example, the engine checks the data type information. If a problem is found, the engine writes a message to the log and stops.

For more information about the effects of changing a CA-Datacom/DB table on existing view descriptors, see Appendix 1, "Information for the Database Administrator," on page 103.

---

## Statements

---

### ASSIGN Statement

**Specifies whether view descriptors that are created from an access descriptor will inherit or select their own SAS column names and formats.**

**Optional statement**

**Applies to:** access descriptor

---

#### Syntax

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

#### Details

The ASSIGN statement specifies whether view descriptors will inherit the SAS column names and formats that were assigned in the parent access descriptor at the time that the access descriptor was created, or whether the column names and formats can be selected in the view descriptor.

If you specify ASSIGN=YES, then default SAS column names and formats are generated for all CA-Datacom/DB field names and these names and formats will be used in all derived view descriptors. You can edit the default column names and formats in the access descriptor with the RENAME, FORMAT, INFORMAT, and CONTENT statements, but you cannot edit them in the view descriptor.

If ASSIGN=NO, which is the default value, default names are not generated and any SAS column names assigned in the access descriptor can be edited in the view descriptor. If you do not specify any column names in the access descriptor, then fields selected in the view descriptor will use default SAS column names and formats, unless you edit them with the RENAME, FORMAT, INFORMAT, and CONTENT statements.

Default SAS column names follow these rules:

- If the CA-Datacom/DB field name is longer than eight characters, SAS uses only the first eight characters. If truncating would result in duplicate names, numbers are appended to the end of the name. For example, the CA-Datacom/DB field names CUSTOMERNAME and CUSTOMERNUMBER would become the SAS column names CUSTOMER and CUSTOME1.
- If the CA-Datacom/DB field name contains invalid SAS name characters, such as a hyphen (-), SAS replaces them with underscores (\_). For example, the CA-Datacom/DB field name FUNC-INT becomes the SAS name FUNC\_INT.
- For a key, the five-character Datacom-NAME is used, if there is one. If that is missing, the first eight nonblank characters of the entity-occurrence name are used.
- For repeating fields, SAS generates unique SAS names by suffixing or overlaying the occurrence number on the last position(s) of the SAS name. For example, the third occurrence of PHONE is PHONE3, the ninth occurrence of LASTNAME is LASTNAM9, and the eleventh occurrence of ADDRESS is ADDRES11. In some cases, this feature causes different fields to have SAS names that differ only in the

suffixed number. For example, if you select BRANCH-NUMBER, BRANCH-PHONE, and BRANCH-ADDRESS and each repeats four times, the SAS names generated by default would be: BRANCH\_1, BRANCH\_2, BRANCH\_3, BRANCH\_4, BRANCH\_5, BRANCH\_6, BRANCH\_7, BRANCH\_8, BRANCH\_9, BRANCH10, BRANCH11, and BRANCH12.

The generated names are not listed in the LIST statement output.

- The SAS name for a compound field contains \*GROUP\*. To see the fully expanded repeating structure, use the LISTOCC statement. To see the field composition, use the LISTINFO statement.
- You can set different default names with a user exit, which is described in Appendix 2, “Advanced Topics,” on page 117.

---

## CONTENT Statement

**Specifies a SAS date format or length.**

**Optional statement**

**Applies to:** access descriptor or view descriptor

---

### Syntax

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

### Details

The CONTENT statement enables you to specify a SAS date format or column length for a CA-Datcom/DB nondate field. A date format means that the CA-Datcom/DB data has the specified representation. The column length determines the number of characters to be accessed.

SAS stores datetime values as the number of days and seconds before and after January 1, 1960. Entering a SAS date format or column length automatically changes the default values for SAS formats and informats. However, if you have previously changed any format or informat values, specifying a CONTENT value does not change those values. Four date formats are used:

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

The *column-identifier* argument can be either the CA-Datcom/DB field name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the access descriptor. If the column contains special characters or national characters, enclose the name in quotation marks.

If you specify ASSIGN=YES when you create an access descriptor, you cannot change the value for this statement when you later create a view descriptor based on that access descriptor. You do not have to issue a SELECT statement for DBMS columns that are named in the CONTENT statement.

SAS supports the CA-Datcom/DB SQL types SQL-DATE, SQL-TIME, and SQL-STMP as binary data. (See “ACCESS Procedure Data Conversions” on page 97 for more information about the default formats that the ACCESS procedure assigns to these DBMS data types.)

---

## 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 file.

To create a descriptor, use a three-level name. The first level identifies the libref of the SAS library where you will store the descriptor. You can store the descriptor in a temporary (WORK) or permanent SAS library. The second level is the descriptor's 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 and view descriptors based on those 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 The CREATE statement for the access descriptor must directly follow the PROC ACCESS statement.
- 2 Database-description statements must follow the CREATE statement: TABLE, TBLSTAT, USER, PASSWORD, DATABASE, DBSTAT, and URT. The order of the database-description statements does not matter.
- 3 The editing statements must follow the database-description statements: ASSIGN, CONTENT, DROP, EXTEND, FORMAT, INFORMAT, KEY, LIST, LISTINFO, LISTOCC, OCCURS, QUIT, RENAME, and RESET. The SELECT and SUBSET statements are used only when creating view descriptors. QUIT is an editing statement but it terminates PROC ACCESS without creating your descriptor.
- 4 The RUN statement is used to signal the end of the ACCESS procedure.

Information from database-description statements is stored in the 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, then you can use the database-description statements in a view descriptor to supply them.

**View descriptors** When you create a view descriptor for an existing access descriptor, you must use the ACCDESC= option with the ACCESS procedure.

When you create view descriptors and access descriptors in the same procedure execution, you must place the statements or groups of statements in the following order:

- 1 You must create an access descriptor before creating a view descriptor based on that access descriptor.
- 2 You should omit the RUN statement from the access descriptor specification.
- 3 Any database-description statements, such as PASSWORD, must precede the editing statements.
- 4 Among the editing statements, RENAME, CONTENT, FORMAT, and INFORMAT can be specified only when ASSIGN=NO is specified in the access descriptor referenced by the view descriptor. The order of the statements within this group usually does not matter; see the individual statement descriptions for any restrictions.
- 5 The RUN statement is used to signal the end of the ACCESS procedure.

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.

If no errors are found, each descriptor is saved when a new descriptor is created or when the RUN statement is processed. If errors are found, error messages are written to the SAS log and processing is terminated. After you correct the errors, resubmit your statements.

For examples of how to create access descriptors and view descriptors, see Chapter 3, “Defining SAS/ACCESS Descriptor Files,” on page 15.

---

## DATABASE Statement

**Identifies the CA-Datcom/DB database to use.**

**Optional statement**

**Applies to:** access descriptor

---

### Syntax

```
DATABASE | DB<=> <">Datcom-database-name<">;
```

### Details

The DATABASE statement enables you to specify the name of the CA-Datcom/DB database that contains the CA-Datcom/DB table you want to access. In CA-Datcom/DB, Database is a 32-character field that names an entity-occurrence of type DATABASE in CA-DATADictionary.

The database name is required only if the table specified in the TABLE statement is not unique in the dictionary. If the name contains special characters or national characters, enclose it in quotation marks.

DB is the alias for the DATABASE statement.

---

## DBSTAT Statement

Indicates the status or version of the specified CA-Datcom/DB database.

Optional statement

Applies to: access descriptor or view descriptor

---

### Syntax

```
DBSTAT<=> <">PROD<"> | <">TEST<"> | <">test-version<">;
```

### Details

The DBSTAT statement enables you to indicate the CA-DATADictionary status and version of the CA-Datcom/DB database that you want to access. The DBSTAT statement is required only if the database you want to use is not the one in production status.

The DBSTAT statement can take one of the following arguments:

PROD	indicates the database that is currently in production. This is the default.
TEST	indicates the database that is currently in test.
<i>test-version</i>	indicates a specific test version of the database. This argument must be in the form of Txxx, where xxx is a 3-digit number.

Other status values, such as HIST, are not used.

---

## DROP Statement

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

Optional statement

Applies to: access descriptor

---

### Syntax

```
DROP <">column-identifier-1<"> <... <">column-identifier-n<">>;
```

## Details

The DROP statement drops the specified DBMS 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.

The *column-identifier* argument can be either the CA-Datcom/DB field 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 the column name in the RESET statement. However, doing so also resets all of the column's attributes (such as SAS column 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 descriptor or view descriptor

---

### Syntax

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

## Details

The EXTEND statement lists information about the DBMS columns in a descriptor. The word \*GROUP\* is displayed to indicate the existence of a group.

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

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 of the DBMS columns in the file, the positional equivalents, the SAS names, the SAS informats, the database contents, the number of occurrences, and the DBMS column types (Alpha or Zoned). 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 informats, the database contents, number of occurrences, DBMS column types, any subsetting clauses, and the word **\*SELECTED\***. Any DBMS 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's SAS name, its positional equivalent, its SAS informat, the database content, number of occurrences, DBMS column type, 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 CA-Datcom/DB field 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 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 the SAS format for a DBMS column.**

**Optional statement**

**Applies to:** access descriptor or view descriptor

---

**Syntax**

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

**Details**

The FORMAT statement changes a SAS column format from its default format; the default SAS column format is based on the data type of the DBMS column. (See "ACCESS Procedure Data Conversions" on page 97 for information about the default formats that the ACCESS procedure assigns to your DBMS data types.)

The *column-identifier* argument can be either the CA-Datcom/DB field 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 CA-Datcom/DB field 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.  $\Delta$

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

---

### Syntax

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

### Details

The INFORMAT statement changes a SAS column informat from its default informat; the default column informat is based on the data type of the DBMS column. (See "ACCESS Procedure Data Conversions" on page 97 for information about the default informats that the ACCESS procedure assigns to your DBMS data types.)

The *column-identifier* argument can be either the CA-Datcom/DB field 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 DBMS 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. △

INFMT is the alias for the INFORMAT statement.

---

## KEY Statement

**Specifies a key field that governs the order that records are read.**

**Optional statement**

**Applies to:** access descriptor or view descriptor

---

### Syntax

**KEY**<=> <">*Datacom-short-name*<">;

### Details

The KEY statement specifies the CA-Datacom/DB short name for a Default Key in the CA-Datacom/DB table. The Default Key value governs the order in which records are read. The Default Key is an optional key that defaults to the Native Key. The Native Key governs how records are stored and maintained.

You can specify another key as the Default Key if you want the records processed in a certain order, but you do not want to specify a SORT clause to achieve that result. You can also specify a Default Key with the DDBKEY= data set option when you execute a SAS procedure.

If the CA-Datacom/DB short name contains special characters or national characters, enclose the name in quotation marks.

---

## LIST Statement

**Lists columns in the descriptor and gives information about them.**

**Optional statement**

**Applies to:** access descriptor or view descriptor

---

## Syntax

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

## Details

The LIST statement lists the 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.

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 column names, and the SAS 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 column 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 column 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 CA-Datcom/DB field 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

---

## 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.

The LISTINFO statement is especially helpful for key fields. It shows the CA-Datcom/DB short name as well as all the columns and levels that make up the key.

The LISTINFO statement can take one of the following arguments:

### ALL

lists the field composition of all the DBMS columns in the file.

### VIEW

lists the field composition of the DBMS columns selected for the view descriptor.

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 field composition of the specified DBMS columns. If the column name contains special characters or national characters, enclose the name in quotation marks.

The *column-identifier* argument can be either the CA-Datcom/DB field 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 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 repeating data 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 occurrences for the specified repeating fields along with information such as the CA-Datcom/DB level, the SAS column name, the occurrence number, the SAS column format and informat, the DB content, 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.

The LISTOCC statement takes the following argument:

*column-identifier*

can be either the CA-Datcom/DB field 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;
```

If the DBMS column name contains special characters or national characters, enclose the name in quotation marks. The column-identifier must be a repeating field.

## OCCURS Statement

**Modifies the occurrences of a repeating data field.**

**Optional statement**

**Applies to:** access descriptor or view descriptor

### Syntax

```
OCCURS <">column-identifier<">
  CONTENT <">occurrence-1<"> <=> SAS-date-format | length
  <... <">occurrence-n<"><=> SAS-date-format | length>;
  |
  DROP occurrence <<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>;
  |
  RENAME <">occurrence-1<"> <=> SAS-name
  < ... <">occurrence-n<"> <=> SAS-name>;
  |
  RESET occurrence-1 <<TO> ... occurrence-n>;
  |
  SELECT occurrence <<TO> ... occurrence-n>;
```

## Details

You use the OCCURS statement to modify values for occurrences of a repeating data field. The OCCURS statement can be used when creating an access descriptor or a view descriptor.

The OCCURS statement enables you to complete the following tasks:

- 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 column name for one or more occurrences.

You can see the two-character numeric level of a CA-Datcom/DB field by using one of the LIST statements. The LVL column displays the word KEY for keys, the number 01 for simple fields and top-level compound fields, 02 for secondary fields, and so on. This listing is for information only and cannot be edited.

The column-identifier must be a repeating field, and can be the CA-Datcom/DB field name or the positional equivalent from the LIST statement. The occurrence argument can be the occurrence name or the occurrence number. If the CA-Datcom/DB field 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 OCCURS 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 CA-Datcom/DB tables and records as changing the DB content attribute of a column. See "CONTENT Statement" on page 68 for more information. For example, if the FIRSTORDERDATE column in the CUSTOMERS table is a repeating field, and you want to change the DB content attribute for occurrences nine and ten, submit the following statement:

```
occurs firstorderdate content 9 yymmdd6. 10 = 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 BRANCHOFFICE column in the CUSTOMERS table, submit the following statement:

```
occurs "BRANCHOFFICE" drop 1 2 3;
```

or

```
occurs "BRANCHOFFICE" 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 `BRANCHOFFICE` column in the `CUSTOMER` table, submit the following statement:

```
occurs "BRANCHOFFICE" format 9 $21. 10 = $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 `BRANCHOFFICE` column in the `CUSTOMERS` table is a repeating field, and you want to change the informat attribute for occurrences nine and ten, submit the following statement:

```
occurs "BRANCHOFFICE" informat 9 $21. 10 = $8.;
```

**RENAME**

enables you to rename a SAS column 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 column name can be renamed. If you set `ASSIGN=YES`, the SAS column name can be renamed in the access descriptor but not in the view descriptor.

You can rename the SAS column 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 `CUSTOMERS` table, submit the following statement:

```
occurs "BRANCH-OFFICE" rename 9 london 10 = 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 83 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 `BRANCHOFFICE` column in the `CUSTOMERS` table, submit the following statement:

```
occurs "BRANCHOFFICE" reset 1 2 3;
```

or

```
occurs "BRANCHOFFICE" 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 BRANCHOFFICE column in the CUSTOMERS table, submit the following statement:

```
occurs "BRANCHOFFICE" select 1 2 3;
```

or

```
occurs "BRANCHOFFICE" select 1 to 3;
```

The result of selecting a key that consists of multiple fields is always a SAS character column. The value of the SAS column is the concatenated values of the component fields. If any of the component fields are numeric, they are converted to character representation, with a format and length set by the interface view engine. The character-only restriction exists so that the key can be used in a WHERE clause.

**PASSWORD Statement**

Specifies a CA-DATADictionary password.

Optional statement

Applies to: access descriptor

**Syntax**

```
PASSWORD | PASS | PW<=> <">Datacom-password<">;
```

**Details**

The PASSWORD statement enables you to supply a CA-DATADictionary password. Not every user ID requires a password.

The value is the 12-character PASSWORD attribute of the PERSON entity-occurrence identified in User Name. If you enter a value, it is saved (in encrypted form) in the access descriptor and in any view descriptor created from it.

If the password contains any special characters or national characters, enclose it in quotation marks.

PASS and PW are aliases for the PASSWORD statement.

**QUIT Statement**

Terminates the procedure.

Optional statement

**Applies to:** access descriptor or view descriptor

---

## Syntax

**QUIT** | **EXIT**;

## 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 column name.**

**Optional statement**

**Applies to:** access descriptor or view descriptor

---

## Syntax

```
RENAME <">column-identifier-1<"> <=> SAS-name
      <...<">column-identifier-n<"> <=> SAS-name>;
```

## Details

The RENAME statement enters or modifies the SAS column name that is associated with a DBMS column. The RENAME statement can be used when creating an access descriptor or a view descriptor. However, the value of the ASSIGN statement affects when the RENAME statement can be used.

When you create an access descriptor, the default setting for a SAS column name is a blank. When ASSIGN=YES, default SAS column names are generated and these SAS column names are used by all of the view descriptors derived from this access descriptor. You can use the RENAME statement to edit the SAS column names assigned in the access descriptor and these renamed SAS column will be used by its view descriptors, unless a RESET statement or another RENAME statement is used in the access descriptor.

If you omit the ASSIGN statement or specify it with a **NO** value, you can use the RENAME statement to assign a SAS column name. In this case, the SAS column names that you enter in the access descriptor will be retained by any view descriptors derived from this access descriptor; however, you can edit them in the view descriptor with the RENAME statement. Column names renamed in the view descriptor apply only to that view descriptor.

The *column-identifier* argument can be either the CA-Datcom/DB field 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 column names that are associated with the seventh DBMS column and the nine-character FIRSTNAME DBMS 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 column name on the right side. The equal sign (=) is optional. If the CA-Datcom/DB field 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 column name 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

---

### Syntax

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

### Details

The RESET statement resets either the attributes of all the DBMS columns or the attributes of the specified DBMS 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.

**Access descriptors** When you create an access descriptor, the default setting for a SAS column name is a blank. However, if you have previously entered or modified any of the SAS column names, the RESET statement resets the modified names to the default names that are generated by the ACCESS procedure. How the default SAS column 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 CA-Datcom/DB field name.

The current SAS column format and informat are reset to the default SAS format and informat, which was determined from the DBMS column's data type. The current DB content is also reset to the default value. 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 column and then select it again within the same procedure execution, the SAS column name, format, informat, and database content 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 cannot be used to restore the default column names and formats in the view descriptor, but it will affect the SELECT statement for the view.

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 CA-Datcom/DB field 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 CA-Datcom/DB field 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.

---

## SELECT Statement

**Selects DBMS columns for the view descriptor.**

**Required statement**

**Applies to:** view descriptor

---

### 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.

The SELECT statement can take one of the following arguments:

#### ALL

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

*column-identifier*

can be either the CA-Datacom/DB field 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 CA-Datacom/DB field name contains special characters or national characters, enclose the name in quotation marks. You can select as many DBMS 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.

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 a View Descriptor” on page 89. For more information about the SORT clause, see “SORT Clause in a View Descriptor” on page 95. You can use either SAS column names or DBMS column names in your selection criteria. Specify your WHERE clause and SORT clause by using the same or separate SUBSET statements. For example, you can submit the following SUBSET statements:

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

---

## TABLE Statement

Indicates the CA-Datacom/DB table you want to use.

Required statement

Applies to: access descriptor

---

### Syntax

```
TABLE<=> <">Datacom-table-name<">;
```

### Details

The TABLE statement specifies the CA-Datacom/DB table that you want to access. *Datacom-table-name* is the 32-character field that names an entity-occurrence of type RECORD in the CA-DATADictionary. (For CA-Datacom/DB R8, the type is TABLE.)

The TABLE statement is required to create an access descriptor and the table name must be unique. If the table name is not unique, you can combine the TABLE statement with the DATABASE, DBSTAT, and TBLSTAT statements until a unique combination is identified.

If the table name contains special characters or national characters, enclose the name in quotation marks.

---

## TBLSTAT Statement

Indicates the status or version of the specified CA-Datacom/DB table.

Optional statement

Applies to: access descriptor or view descriptor

---

### Syntax

```
TBLSTAT<=> <">PROD<"> | <">TEST<"> | <">test-version<">;
```

### Details

The TBLSTAT statement enables you to indicate the CA-DATADictionary status and version of the CA-Datacom/DB table you want to access. The TBLSTAT statement is required only if the database you want to use is not the one in production status.

The TBLSTAT statement can take one of the following arguments:

PROD	indicates the table that is currently in production. This is the default.
TEST	indicates the table that is currently in test.
<i>test-version</i>	indicates a specific test version of the database. This argument must be in the form of Txxx, where xxx is a 3–digit number.

Other status values, such as HIST, are not used.

---

## 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 (edit). The descriptor can exist in either a temporary (WORK) or permanent SAS 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.  $\Delta$

To update a descriptor, use its three-level name. The first level identifies the libref of the SAS 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 statement must be the first statement after the PROC ACCESS statement (Note: The ACCDESC= parameter cannot be specified on the PROC ACCESS statement).

The following statements are not supported when using the UPDATE statement: ASSIGN, RESET, SELECT, and OCCURS 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.  $\Delta$

The following example updates the access descriptor MYLIB.ORDERS on the CA-Datcom/DB table ORDER. In this example, the SAS column names are changed and formats are added.

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

The following example updates an access descriptor MYLIB.EMPLOYEE on the CA-Datcom/DB table EMPLOYEEES and then re-creates a view descriptor VLIB.EMPS, which was based on MYLIB.EMPLOYEE. The original access descriptor included all of the DBMS columns in the table. 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.EMPS must be re-created in order to omit the SALARY and BIRTHDATE columns.

```
proc access dbms=Datacom;
  /* update access descriptor */
  update mylib.employee.access;
  drop salary birthdate;
  list all;

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

---

## URT Statement

**Identifies the user requirements table to use.**

**Optional statement**

**Applies to:** access descriptor or view descriptor

---

### Syntax

URT<=> <">User-Requirements-Table-name<">;

## Details

The URT statement identifies the user requirements table (URT) to be used by the interface view engine when it opens a view descriptor. CA-Datcom/DB requires a URT to open a table. For more information, see “User Requirements Table (URT)” on page 110. However, this statement is optional. If you do not specify a URT when creating an access descriptor or a view descriptor, the engine will create a default URT. A URT name can also be provided with a data set option (see “Data Set Options” on page 117).

If you specify a URT when creating an access descriptor, the interface view engine will use it when it opens any view descriptors created from this access descriptor.

If the URT name contains special characters or national characters, enclose the name in quotation marks.

---

## USER Statement

**Specifies a CA-DATADictionary user ID.**

**Required statement**

**Applies to:** access descriptor

---

### Syntax

**USER**<=> <">*authorized-Datcom-userid*<">;

## Details

The USER statement supplies a required CA-DATADictionary user ID. The user ID is a 32-character entity-occurrence name of a PERSON entity in CA-DATADictionary, which is not necessarily the same as the user's TSO ID.

The value entered in the USER statement is saved in the access descriptor and in any view descriptor created from the access descriptor. The user name and optional password must have retrieval authority on six entity-types: DATABASE, FILE, RECORD, ELEMENT, KEY, and FIELD.

If the user ID contains special characters or national characters, enclose it in quotation marks.

---

## WHERE Clause in a View Descriptor

---

### View WHERE Clause Syntax

You can use a WHERE Clause in a view descriptor to select specific records from a CA-Datcom/DB table. You can reference any CA-Datcom/DB field included in the view descriptor.

A WHERE clause in a view descriptor consists of the word WHERE followed by one or more conditions that specify criteria for selecting records from one CA-Datcom/DB table. (WITH and WH are valid aliases for the word WHERE.)

A condition can be one of the following:

```
field-name<(occurrence)>|key-name operator value
field-name* operator field-name*
field-name<(occurrence)>|key-name range-operator low-value * high-value
```

The user-supplied elements of the WHERE clause conditions are described here.

**field-name<(occurrence)>|key-name**

is the CA-Datcom/DB name of the field or key for which you are specifying criteria. The field must be selected in the view descriptor. The interface view engine assumes that the name in a condition is a SAS name. If it is not, the name will be treated as a CA-Datcom/DB name.

If the field is a repeating field, you must specify the occurrence of that field in parenthesis, where occurrence is one of the following:

*n* indicates the *n*th occurrence. For example,

```
where address(3) contains dallas
```

selects those records where the third occurrence of ADDRESS contains DALLAS.

ALL indicates all occurrences selected in the view descriptor. For example, the WHERE clause below selects those records where all occurrences of ADDRESS contains DALLAS.

```
where address(all) contains dallas
```

ANY indicates any occurrence. An asterisk (\*) can be used instead of ANY. For example,

```
where address(any) contains dallas
```

selects those records where any occurrence of ADDRESS contain DALLAS. You could have used ADDRESS(\*) instead.

*operator*

is one of the following:

= or EQ	equal to
> or GT	greater than
< or LT	less than
!= or $\neq$ or NE	not equal
>= or GE or GTE	greater than or equal to
<= or LE or LTE	less than or equal to
CONTAINS or CONTAINING	contains
$\neg$ CONTAIN or $\neg$ CONTAINING	does not contain
!CONTAIN or !CONTAINING	does not contain

*range-operator*

is one of the following:

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

!= or  $\neq$  or NE is outside the range

*value, high-value, and low-value*

represent valid values for the field or key.

For more information, see “Specifying Values in View WHERE Clauses” on page 92.

## The Asterisk in View WHERE Clauses

The asterisk (\*) is required when comparing two field names. For example, the following WHERE clause selects those records where the wages are less than the commission:

```
where ytd-wages*<ytd-commission*
```

This WHERE clause

```
where ship-quant*=order-quantity*
```

selects those records where the ship-quantity is equal to the order-quantity.

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

```
where number spans 2300*2400
```

The WHERE clause

```
where lastname spans 'A'*'Smith'
```

selects those employees with last names up to Smith. See “Character Fields in View WHERE Clauses” on page 92 for details on the use of quotation marks.

If the asterisk appears in a value, enclose the value in quotation marks or use the DDBSPANS system option to specify another special character. For more information about system options, see “System Options for the CA-Datacom/DB Interface” on page 113.

## View WHERE Clause Expressions

Conditions can be combined 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.

```
where cost=.50 & (type=ansil2 | class=sorry)
```

Conditions can also be preceded by NOT (X).

```
where cost=.50 & not (type=ansil2 | class=sorry)
```

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

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

The next WHERE clause selects all records where PART is 9846 and ON-HAND is greater than  $2 \times 10^6$ :

```
where part=9846 & on-hand>2.0e+6
```

---

## Specifying Values in View WHERE Clauses

### Character Fields in View WHERE Clauses

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,

```
where lastname=Smith
```

extracts data for SMITH, and the next example extracts data for Smith:

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

If the value is shorter than the field, it is padded on the right with blanks before the comparison. (No padding is done if you use the CONTAINS operator.) If the value is longer than the field, it is truncated to the field length before the comparison is done. The WHERE clause

```
where name=Anderson
```

selects all records where NAME is ANDERSON. The WHERE clause

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

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.

In this example, either of these WHERE clauses

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

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

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 quotation mark. You can also use double quotation marks around a value. Also, two consecutive double quotation marks become one double quotation mark if surrounded by double quotation marks. If two consecutive double quotation marks are surrounded by single quotation marks, they remain two double quotation marks and conversely.

### Date Values in View WHERE Clauses

You can use the DB Content statement to specify a date format. Using this statement, you can specify the dates according to your SAS informat. Do not use 'd' as you would for SAS software.

### \$HEX. Format Fields in View WHERE Clauses

For fields that are converted to \$HEX. format because of their data type or length (see "ACCESS Procedure Data Conversions" on page 97), the value must be specified in hexadecimal. A value longer than the field is truncated to the field length before the comparison is done. A value shorter than the field is padded on the right with binary zeros before the comparison. For example, if CODE has \$HEX4. format,

```
where code=f1f
```

extracts the data for CODE equals 10 (F1F0).

## Values That Do Not Fit the Field Picture

If you specify a value that does not fit the field's picture, you might receive an error, or the value might be adjusted to fit the picture before sending the request to CA-Datcom/DB.

The following examples illustrate how various misfit values are handled. Assume throughout that COST has a database length of 5, with 2 decimals.

In the first set of examples, some misfit values produce errors, some are truncated, and some cause operators to be changed. Errors occur when the equals operator or not equals operator is used with a misfit value. Operators are changed when that change plus truncation means the value will fit the picture and still produce the results you intended.

**Table 6.2** Various Misfit Field Values

Condition	Request Sent to CA-DATACOM/DB	
cost=.003	Error	(underflow: field has two decimals)
cost>.003	cost>0.00	(truncated)
cost>3.0052	cost>3	(truncated)
cost<.0001	cost $\leq$ 0.00	(truncated, < changed to $\leq$ )
cost<20.001	cost $\leq$ 20	(truncated, < changed to $\leq$ )

The next examples show values that exceed the field size. If possible, your values are replaced with the largest value that can be stored in the field.

**Table 6.3** Field Values That Are Too Large

Condition	Request Sent to CA-DATACOM/DB
cost<11123	cost $\leq$ 999.99
cost $\neq$ 9999	Error (overflow, field cannot store integers > 999)
cost $\geq$ -12345	cost $\geq$ - 999.99

## Masking Values in View WHERE Clauses

When a condition includes the EQ, NE, CONTAINS, or NOT CONTAINS operator and the field is in display code, you can mask the value. That is, you can specify that only certain positions within the value are to be compared to those positions in the field. A pound sign (#) marks the positions that you do not want to be compared. For example,

```
where zipcode eq 7#8
```

selects all records with zip codes that have a 7 in the first position and an 8 in the third position. The condition

```
where lastname contains m#n
```

selects all records with last names such as Mendoza, Harman, and Warminsky.

If you use the EQ or NE operators and you mask a value that is shorter than the database field, your values are padded on the right with mask characters. (No padding is done for NOT CONTAINS.) For example,

```
where lastname eq m#n
```

would select records with last names such as Mendoza, McNeal, and Monroe. Names such as Harman or Warminsky would not qualify.

Use the DDBMASK system option to change the default masking character (#). For more information about system options, see “System Options for the CA-Datcom/DB Interface” on page 113.

## Multi-Field Keys in View WHERE Clauses

For a condition that specifies a multi-field key, you might need to enclose each value with delimiters.

*Note:* You cannot use compound fields in the WHERE clause.  $\Delta$

For multi-key fields, use a delimiter character\* before and after each value if the value you are entering is not the same length as the multi-field key and you are using either NOT CONTAINS or the mask character. Values for keys are always in display code. For example, suppose INIT-ID is a multi-key field. INIT is a character field of length 3, and ID is a numeric field of length 7. The WHERE clause

```
where init-id=\jde\27#\
```

selects all records where the initials are JDE and the ID number starts with 27. Your value for ID is padded on the right with mask characters, so the entire value is treated as if you had specified JDE27#####.

You can omit delimiters if you specify the same number of characters as the multi-field key contains. For example, this WHERE clause

```
where init-id=jde27#####
```

also selects all records where the initials are JDE and the ID number starts with 27, just as in the previous example. No delimiters are required here because JDE27##### is 10 characters long, which is the same size as the key field.

When you do not include delimiters or masked characters in the value, blanks or zeros are used for padding. The WHERE clause

```
where weight-sex=78m
```

selects all records where weight equals 78 and sex equals M. The value is treated as if it had been specified as \78\m\.

On the other hand, the WHERE clause

```
where age-degree=25bs
```

selects all records where age equals 25 and degree equals BS. The value is treated as if it had been specified as \25\bs \.

*Note:* A considerable amount of processing is required when a procedure must convert an apparently simple condition into a complex request to CA-Datcom/DB. For example, if the fields AGE and SEX are not contiguous within the record, the procedure converts the condition AGE-SEX<25M to SEX<M OR (SEX=M AND AGE<25) before submitting the request. CA-Datcom/DB, in turn, processes the request and, if possible, uses permanent indexes to satisfy it.  $\Delta$

## Guidelines for View WHERE Clauses

Consider the following guidelines when you specify a WHERE clause in the view descriptor:

- You can enter a WHERE clause or a SORT clause or both, in either order. But if you enter both, do not use a terminator between them.

---

\* Use the DDBDELIM system option to change the default delimiter character (\). For more information about system options, see “System Options for the CA-Datcom/DB Interface” on page 113.

- The keyword WHERE is not required unless the WHERE clause is the second clause (following the SORT clause). The SORT clause must begin with SORT.
- CA-Datcom/DB does not have a date data type. However, the selection criteria will honor a SAS date format if you specify one in the CONTENT and INFORMAT statements.
- The CA-Datcom/DB fields must be selected in the view descriptor in order for you to use them in the WHERE clause.
- All conditions in the WHERE clause must refer to fields in a single table. To join conditions that pertain to two CA-Datcom/DB tables, use the SQL procedure.
- If you enter a SAS WHERE clause when you use the view descriptor in a SAS procedure, the SAS WHERE clause is connected to the WHERE clause in the view descriptor (if any) with the AND operator.
- The WHERE clause is not parsed (or checked) until the interface view engine tries to execute it for a procedure.
- Field names in the WHERE clause conditions can be SAS names or CA-Datcom/DB names. However, you should use SAS names for repeating fields or for fields within repeating fields.
- Character literals and values for zoned decimal fields can contain the pound sign (#) to indicate masking out characters for pattern matching operations.

For more information about specifying WHERE clauses, see “Deciding How to Specify Selection Criteria in CA-Datcom/DB” on page 126.

---

## SORT Clause in a View Descriptor

---

### Overview of the SORT Clause

When you define a view descriptor, you can also include a SORT clause to specify data order. You can reference only the CA-Datcom/DB fields selected for the view descriptor.

Without a SORT clause or a SAS BY statement, the data order is determined by the Native Key for the CA-Datcom/DB table (or by the Default Key specified in the access or view descriptor).

A SAS BY statement automatically issues a SORT clause to CA-Datcom/DB. However, the SAS BY statement might cause grouping of the output results in some procedures; this might not be what you want.

If a view descriptor already contains a SORT clause, the BY statement overrides the SORT clause 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.

---

### View SORT Clause Syntax

The syntax for the SORT clause is

```
SORT field-name <ASCENDING|UP|A> <DESCENDING|DOWN|D>
    <,<field-name...>
```

The elements of the SORT clause are described here.

*field-name*

is a CA-Datcom/DB field name or SAS column name of a CA-Datcom/DB field included in the view descriptor. Use commas to separate sort keys. You can also specify either ascending or descending order for each field name.

ASCENDING | UP | A

specifies that you want the data ordered by ascending values of the *field-name*. ASCENDING is the default.

DESCENDING | DOWN | D

specifies that you want the data ordered by descending values of the *field-name*.

If you specify more than one CA-Datcom/DB field, the values are ordered by the first named field, then the second, and so on.

## View SORT Clause Example

The following SORT clause causes the values to be presented in ascending order based on the values in field STATE, then within states in descending order based on the values in field CITY:

```
sort state, city down
```

## View SORT Clause Guidelines

Consider the following guidelines when you specify a SORT clause in the view descriptor:

- You can enter a WHERE clause or a SORT clause or both, in either order. But if you enter both, do not use a terminator between them.
- The keyword WHERE is not required unless the WHERE clause is the second clause (following the SORT clause). The SORT clause must begin with SORT.
- If you specify a SAS BY clause when you execute a procedure, it replaces the SORT clause in the view descriptor. However, if the SAS procedure includes the NOTSORTED option, the SAS BY clause is ignored and the SORT clause in the view descriptor is used. A message is written to the LOG window when the NOTSORTED option causes a SORT clause to be ignored.
- The CA-Datcom/DB fields must be selected in the view descriptor in order for you to use them in the SORT clause.
- In the SORT clause, you can specify multiple fields, separated by commas.
- The SORT clause is not parsed (or checked) until the interface view engine tries to execute it for a procedure.
- Field names in the SORT clause conditions can be SAS names or CA-Datcom/DB names. However, you should use SAS names for repeating fields or for fields within repeating fields.

## Creating and Using View Descriptors Efficiently

When creating or using view descriptors, follow these guidelines to minimize the use of CA-Datcom/DB and your operating system resources and to reduce the time CA-Datcom/DB takes to access data.

- Select only the fields your program needs. Selecting unnecessary fields adds extra processing time.
- Specify the order in which records are presented to SAS (with a SORT clause or a SAS BY statement) only if SAS needs the data in a particular order for subsequent processing.

The SAS BY statement issues an ordering clause to CA-Datcom/DB so that CA-Datcom/DB does the sorting using its system resources. This SORT clause overrides any existing SORT clause for the view descriptor. If you decide to use a SORT clause or a SAS BY statement, order by a key, which is indexed, when possible. (For help in determining which fields in a table are indexed, see your DBA or the table's creator.)

As an alternative to using a SORT clause, which consumes CPU time each time you access the CA-Datcom/DB table, you could use the SORT procedure with the OUT= option to create a sorted SAS data file. This is a better approach for data you want to use many times.

- If a view descriptor describes a large CA-Datcom/DB table and you will use the view descriptor often, it might be more efficient to extract the data and place them in a SAS data file. (Even though the extracted data file will be very large, you will need to create it only once. Also, the extracted data will not reflect any subsequent updates to the table.) See "Performance Considerations" on page 41 for more information about when it is best to extract data.
- Specify selection criteria to retrieve a subset of the records CA-Datcom/DB software returns to SAS, where possible.
- If you use a Default Key, the interface view engine will use an index read instead of a sort if it can. Index reads are faster, but not always possible. For example, an index read is not possible if you specify multiple sort keys, multiple WHERE clause conditions, or a WHERE clause condition with a column that is not a key.
- Omit the KEY statement if you do not need a certain order and you want to retrieve the data sequentially. Otherwise, you might cause an unnecessary sort. PROC FSBROWSE, FSEDIT, and FSVIEW automatically use random access and require a value in the Default Key field.
- You can provide your own URT that is fine-tuned for your applications.

---

## ACCESS Procedure Data Conversions

The following table shows the default formats that SAS assigns to each CA-Datcom/DB data type. The default formats also become the default informat. *len* is the value of the LENGTH attribute of the CA-Datcom/DB field. *dec* is the value of the DECIMALS attribute of the CA-Datcom/DB field.

**Table 6.4** Default SAS Column Formats for CA-Datcom/DB Data Types

Field Type	Field Description	Default SAS Format
C	character	$\$len.$
B	binary:	
	for length $\leq 8$ , unsigned	$(2xlen+1).dec$
	for length 8, signed	$(2xlen+2).dec$
	for length $> 8$	$\$HEX(2xlen).$
	for length = 4, semantic-type= SQL-DATE	DATE9.

Field Type	Field Description	Default SAS Format
	for length = 3, semantic-type= SQL-TIME	TIME8.
	for length = 10, semantic-type= SQL-STMP	DATE TIME30.6
D	packed decimal:	
	for length $\leq 16$ , unsigned	$(2 \times len + 1).dec$
	for length 16, signed	$(2 \times len + 2).dec$
	for length $> 16$	$\$HEX(2 \times len).$
E	extended floating-point	$\$HEX(2 \times len).$
G	graphics data	$\$HEX(2 \times len).$
H	hexadecimal character	$\$len.$
K	kanji (same as Y)	$\$HEX(2 \times len).$
L	long floating-point	E24.
N	numeric (zoned decimal):	
	for length 16, unsigned	$len.dec$
	for length 16, signed	$(len + 1).dec$
	for length $> 16$	$\$HEX(2 \times len).$
S	short floating-point	E14.
T	PL/I bit representation	$\$HEX(2 \times len).$
Y	double-byte character set (DBCS)	$\$HEX(2 \times len).$
Z	mixed DBCS and single byte	$\$HEX(2 \times len).$
2	halfword binary (aligned), unsigned	5.dec
2	halfword binary (aligned), signed	6.dec
4	fullword binary (aligned), unsigned	9.dec
4	fullword binary (aligned), signed	10.dec
8	doubleword binary (aligned), unsigned	17.0

Note that CA-Datcom/DB numeric fields are copied into SAS character columns with a  $\$HEX.$  format if they are too long to fit in a SAS numeric column. For example, a CA-Datcom/DB field of type B with a length of 30 is copied into a SAS column with  $\$HEX60.$  format. A field of type B with a length of 5 and *dec* of 2 is copied into a SAS column with 11.2 format. An error message appears if any precision is lost.

The maximum SAS format width is 200, so SAS uses 200 for CA-Datcom/DB fields whose length exceeds 200.

You might want to change the default format whenever it does not seem appropriate for the values stored in the table. For example, a packed decimal field of length 4 and 2 decimal places would have a default SAS format of 7.2. A very large negative number with a decimal (such as -99,999.99) might not fit.

If SAS software discovers the output format is too small, it issues the following warning message to the error log: AT LEAST ONE W.D FORMAT WAS TOO SMALL FOR THE NUMBER TO BE PRINTED. THE DECIMAL POINT MIGHT BE SHIFTED BY THE BEST FORMAT. The message can occur, for example, when you invoke the PRINT procedure. If this message appears, you should specify a larger width.

The format determines how values in the SAS column are displayed; it does not affect how those values are stored. Their storage is determined by their

CA-Datcom/DB type and length. Therefore, you cannot replace a character format with a numeric format or conversely.

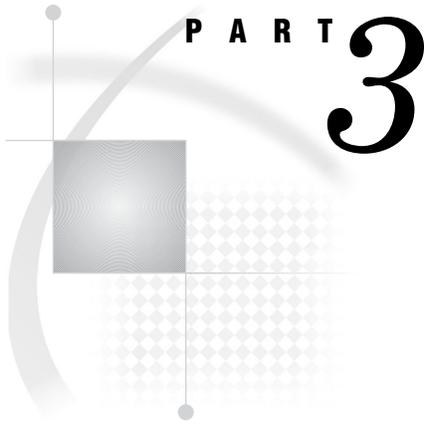
If numeric values in the table are a lot smaller than their length implies, space on the output print line can be conserved by specifying smaller *w.* or *w.d* formats.

Each key is converted to one SAS character column, even if the key is numeric or has more than one component field. The length of a key becomes its default format width. You cannot change the format for a key.

If you assign a date format to a numeric field, be sure that you also specify the SAS date format in the DB Content field to indicate you are storing dates in your table.

For binary data types, if the SEMANTIC-TYPE attribute is not set to SQL-DATE, SQL-TIME, nor SQL-STMP, the data will be treated as normal binary data. To store a SAS date for a normal binary field, use the ACCESS procedure CONTENT statement to assign a date format.

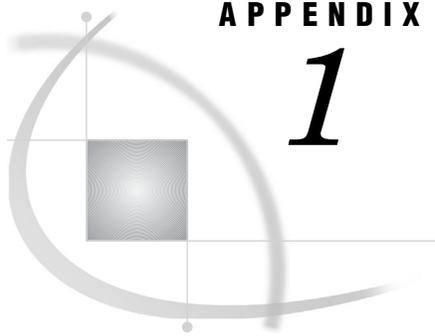




## Appendixes

- Appendix 1* . . . . . **Information for the Database Administrator** 103
- Appendix 2* . . . . . **Advanced Topics** 117
- Appendix 3* . . . . . **Data and Descriptors for the Examples** 129
- Appendix 4* . . . . . **Recommended Reading** 153





## APPENDIX

## 1

## Information for the Database Administrator

---

<i>Introduction to the Information for the Database Administrator</i>	103
<i>How the SAS/ACCESS Interface to CA-Datcom/DB Works</i>	104
<i>Overview for the Database Administrator</i>	104
<i>Using the CA-Datcom/DB Interface View Engine</i>	104
<i>How the CA-Datcom/DB Interface View Engine Works</i>	104
<i>Calls Made on Behalf of the ACCESS Procedure</i>	104
<i>Calls Made by Other SAS Procedures</i>	105
<i>Retrieval Processing</i>	105
<i>Retrievals with a WHERE Clause or SORT Clause</i>	105
<i>Retrievals with No WHERE Clause</i>	105
<i>The Internal Record ID (RID)</i>	106
<i>Update Processing</i>	107
<i>Updating, Deleting, and Adding Data Records</i>	107
<i>Repositioning to an Inserted Record</i>	107
<i>Recovery Processing</i>	108
<i>How Changing the CA-DATADictionary Database Affects Descriptor Files</i>	109
<i>Changes That Affect the Descriptor Files</i>	109
<i>Changes That Do Not Affect Existing View Descriptors</i>	109
<i>Changes That Might Affect Existing View Descriptors</i>	109
<i>Changes That Cause Existing View Descriptors to Fail</i>	110
<i>SAS Security with CA-Datcom/DB</i>	110
<i>User Requirements Table (URT)</i>	110
<i>Locks and the Spool Files</i>	111
<i>Direct Addressing and Access by Row Number</i>	111
<i>Password Encryption/Decryption in CA-Datcom/DB</i>	112
<i>Maximizing the CA-Datcom/DB Interface Performance</i>	112
<i>Multi-Tasking with CA-Datcom/DB</i>	112
<i>Error Messages and Debugging Information for CA-Datcom/DB</i>	113
<i>System Options for the CA-Datcom/DB Interface</i>	113

---

### Introduction to the Information for the Database Administrator

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

---

## How the SAS/ACCESS Interface to CA-Datcom/DB Works

---

### Overview for the Database Administrator

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

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

When you update either an access descriptor or a view descriptor, SAS does not call CA-Datcom/DB or CA-DATADictionary.

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

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

$\triangle$

---

### Using the CA-Datcom/DB Interface View Engine

#### How the CA-Datcom/DB Interface View Engine Works

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

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

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

#### Calls Made on Behalf of the ACCESS Procedure

The ACCESS procedure calls the interface view engine to retrieve information from the CA-DATADictionary database about entity-occurrence names and attributes. The engine sends this information (for example, name, data type, level) to the ACCESS

procedure for each field in the table. The procedure stores this information in the access descriptor for later use when creating view descriptors.

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

## Calls Made by Other SAS Procedures

SAS procedures can access records in a CA-Datcom/DB table 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 (for example, field name, data type, key, level, and occurs specifications) to process CA-Datcom/DB data records as if they were rows in a SAS data file.

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

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

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

---

## Retrieval Processing

---

### Retrievals with a WHERE Clause or SORT Clause

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

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

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

---

### Retrievals with No WHERE Clause

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

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

*Note:* You can also manually set the Default Key to blanks. In this special situation, if you do not specify a WHERE clause or a SORT clause and the CA-Datcom/DB table is opened for retrieval only, the engine avoids accessing any index; it uses GSETP and GETPS commands to read the data area of the table sequentially in its physical sequence. This method is the fastest way to extract an entire table into a SAS file. GETPS and GSETP use look-ahead buffering by blocking records. Therefore, to avoid update contention, these commands are used only for retrieval.  $\Delta$

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

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

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

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

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

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

---

## The Internal Record ID (RID)

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

All retrieval commands return an internal record ID (RID). CA-Datcom/DB sends the RID to the interface view engine. A SAS procedure can request the RID from the engine even though it is internal and not a row number. The engine uses most of the

request area for the RID (approximately 256 bytes), not just the internal seven-byte record-ID. Commands such as REDLE and RDULE can use the retained RID information in a rebuilt request area to reestablish the previous position in the index. Thus, after a record is retrieved, its RID can be used to retrieve the record again.

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

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

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

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

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

## Update Processing

### Updating, Deleting, and Adding Data Records

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

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

For more information about the DDBLOAD option, see “Data Set Options” on page 117 and “System Options for the CA-Datcom/DB Interface” on page 113.

### Repositioning to an Inserted Record

If the Default Key is the Master Key and DUPE-MASTER-KEY is N, repositioning takes place efficiently. Without a WHERE clause, the RDULE command locks the

record. The LOCKX command can locate the index entry just added. Subsequent commands move backward and forward, traversing the actual index.

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

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

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

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

For DDBLOAD=1 and a WHERE clause, the new records are not accessible for viewing since they are not on the original Select File and you suppressed the overhead of keeping track of them. You must either reissue the WHERE clause or exit the procedure and call it again to see the new records.

- $\square$  If DDBLOAD equals 0 (the default), the engine takes the time to keep track of the new records for repositioning. With no WHERE clause, an entry is placed in the index for the Default Key and that will become the new position. With a WHERE clause or the type of SORT clause that causes a Select File, the engine appends the new record ID at the end of the Select File (not in value order).

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

---

## Recovery Processing

The interface view engine does not perform recovery processing as part of its normal behavior. The engine issues a database commit statement only once after a procedure or DATA step completes, unless an error is encountered. When an error is encountered, processing stops and no rollback is attempted. You can implement basic recovery techniques by using the DDBCOMIT= and DDBERLMT= data set options, either individually or together. The DDBCOMIT= option enables you to specify the number of records that are processed before a database commit is issued, including rows that are not processed successfully. DDBERLMT= enables you to specify the number of errors before SAS stops processing and initiates a rollback. For more information about the options, including their interaction with each other, see “Data Set Options” on page 117.

---

## How Changing the CA-DATADictionary Database Affects Descriptor Files

---

### Changes That Affect the Descriptor Files

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

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

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

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

---

### Changes That Do Not Affect Existing View Descriptors

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

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

---

### Changes That Might Affect Existing View Descriptors

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

---

## Changes That Cause Existing View Descriptors to Fail

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

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

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

---

## SAS Security with CA-Datcom/DB

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

- Set up all access descriptors yourself.
- Set up all view descriptors yourself and give them to users on a selective basis.
- Give users read-only access or no access to the SAS library in which you store the access descriptors. Read-only access prevents users from editing access descriptors and enables them to see only the fields selected for each view descriptor. No access prevents users from doing LIST ALL commands to see the contents of the access descriptor.
- Set up several access descriptors for different users.
- Use the DDBUPD systems option to have a read-only system. (For details, see “System Options for the CA-Datcom/DB Interface” on page 113.)

---

## User Requirements Table (URT)

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

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

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

*Note:* ACCESS=SEQ is not used. Use ACCESS=RANSEQ. The engine never alters the type of ACCESS that you specify in a URT. Also, AUTODXC=NO is not used in a URT.  $\Delta$

For more information about URTs, see the appropriate CA-Datcom/DB documentation.

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

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

## Locks and the Spool Files

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

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

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

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

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

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

## Direct Addressing and Access by Row Number

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

For example, if you are on row 3 while someone else is deleting row 4, you go forward to row 5. In another situation, if you update row 3 so that it no longer matches your WHERE clause, it is gone if you ever try to go back to it (even in the same session).

CA-Datcom/DB re-evaluates each retrieval against the WHERE clause and will not return data records that once qualified but currently do not qualify.

---

## Password Encryption/Decryption in CA-Datcom/DB

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

---

## Maximizing the CA-Datcom/DB Interface Performance

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

- Does the program need all the CA-Datcom/DB fields? If not, create and use an appropriate view descriptor that includes only the fields needed.
- Do the selection criteria retrieve only those data records needed for subsequent analysis? If not, specify different conditions so that the selected records are restricted for the program being used.
- Is the data going to be used by more than one procedure in one SAS session? If so, consider extracting the data and placing it in a SAS data file for SAS procedures to use, instead of enabling the data to be accessed directly by each procedure. See "Performance Considerations" on page 41 for circumstances in which extracting data is the more efficient method.
- Do the records need to be in a particular order? If so, include a SORT clause in the appropriate view descriptors or a SAS BY clause in a SAS program.
- Do the selection criteria enable CA-Datcom/DB to use key (indexed) fields and non-indexed fields efficiently? See "WHERE Clause in a View Descriptor" on page 89 for some guidelines on specifying efficient selection criteria.
- Does your WHERE clause cause CA-Datcom/DB to create a temporary index, which often requires excessive processing time? For more information about displaying WHERE clauses and messages about creating temporary indexes, see "DDBTRACE= Data Set Option" on page 122.
- What kind of locking mechanism is required? (See "Locks and the Spool Files" on page 111.)
- Are your CA-DATADictionary elements well-matched to your view descriptors?
- Would a different URT help?
- Would use of a default key give you a faster result?

---

## Multi-Tasking with CA-Datcom/DB

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

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

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

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

---

## Error Messages and Debugging Information for CA-Datcom/DB

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

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

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

The WHERE clause display shows the following:

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

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

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

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

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

---

## System Options for the CA-Datcom/DB Interface

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

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

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

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

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

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

*Note:* In previous releases of the SAS/ACCESS interface to CA-Datcom/DB, DDBUPD was called DDBENGMD.  $\Delta$

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

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

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

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

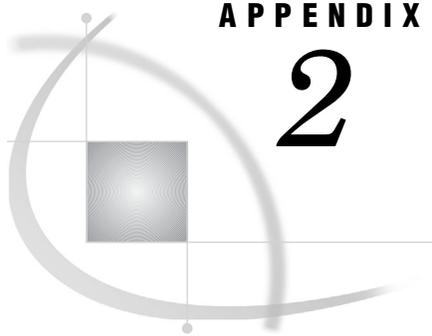
The CA-Datcom/DB system options are listed in the following table.

**Table A1.1** CA-Datcom/DB System Options

Systems Option	Default	Purpose
DDBDBN	blanks	Database name
DDBPW	blanks	Password for CA-DATADictionary
DDBSV	PROD	Status/Version
DDBURT	blanks	User Requirements Table to be used
DDBUSER	blanks	User ID for CA-DATADictionary
DDBDELIM	\	Changes the delimiter
DDBUPD (formerly DDBENGMD)	0	Engine mode (update or read-only)
DDBLOAD	0	Mode for loading data records
DDBLOCK	0	Spooling mechanism
DDBMASK	#	Changes the mask character

<b>Systems Option</b>	<b>Default</b>	<b>Purpose</b>
DDBMISS	blank	Sets missing values to blanks or X'00
DDBSPANS	*	Changes the SPANS character
DDBTASK	2	Number of concurrent tasks
DDBTRACE	0	Displays WHERE clauses and debug traces





## APPENDIX

## 2

## Advanced Topics

---

<i>Introduction to Advanced Topics</i>	117
<i>Data Set Options</i>	117
<i>Using Multiple View Descriptors</i>	124
<i>User Exits from CA-Datcom/DB</i>	124
<i>Deleting and Inserting Data Records in CA-Datcom/DB</i>	124
<i>Missing Values (Nils) in CA-Datcom/DB Tables</i>	125
<i>SAS WHERE Clause Conditions Not Acceptable to CA-Datcom/DB</i>	125
<i>Deciding How to Specify Selection Criteria in CA-Datcom/DB</i>	126
<i>WHERE Clause in the View Descriptor</i>	126
<i>SAS WHERE Clause</i>	126
<i>Validation of Data Values in CA-Datcom/DB</i>	126
<i>Validation against CA-DATADICIONARY</i>	126

---

## Introduction to Advanced Topics

This appendix contains more details about some advanced topics, such as data set options, processing inserts and deletions, multiple views, and multiple windows, as well as how you can optimize your selection criteria. The discussions supplement other portions of this manual.

---

## Data Set Options

Data set options enable you to override some of the run-time options that are stored in view descriptors. There are no LIBNAME options.

Here is a list of available data set options.

DDBCOMIT=*n*

DDBERLMT=*n*

DDBKEY= '*default-key*'

DDBLOAD=0|1

DDBLOCK=0|1

DDBPW= '*password*'

DDBSV= '*status/version*'

DDBTRACE=0|1

DDBURT= '*User-Requirements-Table-name*'

DDBUSER= '*userid*'

You can specify the data set options with the DATA= argument in any PROC statement or DATA step. They are effective for that single execution of the procedure. Data set options will override the corresponding values stored in the view descriptor.

The following example executes the FSEDIT procedure using a view descriptor named CUSTM. The data set options specified in the PROC statement will set the user ID to JOHN and the password to MINE, regardless of what user ID and password was included in the view descriptor.

```
proc fsedit data=vlib.custm (ddbuser='john' ddbpw='mine');
run;
```

A description of each data set option follows.

---

## DDBCOMIT= Data Set Option

**Specifies the number of rows that will be processed before a database COMMIT statement is issued.**

**Default:** 0

**Alias:** DBCOMMIT=

**See also:** DDBERLMT=

### Syntax

**DDBCOMIT= *n***

*n*

is an integer that is equal to or greater than zero.

### Details

The DDBCOMIT= data set option affects update, delete, and insert processing. The number of rows includes rows that are not processed successfully. DDBCOMIT=0 is the default setting and specifies that a commit is issued only once after the procedure or DATA step completes. In the following example, a commit is issued after every 10 rows are inserted:

```
proc append data=mylib.staff base=datacom.dept (ddbload=1 ddbcommit=10);
run;
```

For DDBCOMIT= to be enforced, CA-Datcom/DB logging must be turned on and TXNUNDO=YES must be specified in the User Requirements Table (URT). TXNUNDO=YES is specified in the default URT that is shipped with SAS/ACCESS interface to CA-Datcom/DB software. If DDBCOMIT > 0 and logging is off, then the locks are released, but a commit is not issued.

If DDBCOMIT=0 and DDBERLMT > 1, a rollback is attempted when DDBERLMT= is reached. If DDBCOMIT > 0, a commit can be issued before a rollback that is needed by the DDBERLMT= option. For more information, see DDBERLMT=.

In PROC SQL, the DDBCOMIT= option enables PROC SQL UNDO\_POLICY=REQUIRED for both DDBCOMIT=0 and DDBCOMIT > 0.

## DDBERLMT= Data Set Option

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

Default: 1

Alias: ERRLIMIT=

See also: DDBCOMIT=

### Syntax

**DDBERLMT=** *n*

*n*

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

### Details

For insert, update, delete, and append operations, a rollback is issued when DDBERLMT= is reached. For read operations, processing stops when the specified number of errors occurs. If DDBERLMT is set to 0, no rollback is attempted and processing continues to completion regardless of the number of errors encountered.

For DDBERLMT= to be enforced, CA-Datcom/DB logging must be turned on and TXNUNDO=YES must be specified in the User Requirements Table (URT). TXNUNDO=YES is specified in the default URT that is shipped with SAS/ACCESS interface to CA-Datcom/DB software.

If you specify a value for DDBCOMIT= other than 0, rollbacks affected by the DDBERLMT= option might not include records that are processed unsuccessfully because they were already committed by DDBCOMIT=. The following table summarizes the interaction between DDBERLMT= and DDBCOMIT=.

**Table A2.1** Interaction between the DDBERLMT= and DDBCOMIT= Data Set Options

DDBERLMT=	DDBCOMIT=	Result
1	0	Defaults. No commit or rollback. Processing ends upon the first error.
0	0	No commit or rollback. Processing does not end upon an error.
0	>0	A commit is performed when the transactions that are processed equal DDBCOMIT=. No rollback is performed. Processing does not end upon an error.

DDBERLMT=	DDBCIMIT=	Result
>1	0	No commit. A rollback is performed, processing ends. or both when DDBERLMT= is reached.
>1	>0	A commit is performed when the commit count equals DDBCIMIT=. A rollback is performed when the error counts equals DDBERLMT=. If a multiple of DDBCIMIT= equals DDBERLMT=, then a rollback, not a commit, is performed when DDBERLMT= is reached.

---

## DDBKEY= Data Set Option

Specifies a Default Key.

---

### Syntax

**DDBKEY=**'*default-key*'

'*default-key*'

is a CA-Datcom/DB short name for the key that you want to use.

### Details

The DDBKEY= data set option specifies an optional Default Key for the CA-Datcom/DB table. If one is specified in the view descriptor, this data set option overrides it.

---

## DDBLOAD= Data Set Option

Specifies a fast-loading process.

Default value: 0

---

### Syntax

**DDBLOAD=** 0 | 1

0

causes the software to reposition itself after adding a new record.

1

does not reposition the software after adding new records.

## Details

After a new record is added, the interface view engine attempts to determine its position so it can find and display the record for interactive online updating. The DDBLOAD= data set option enables you to avoid the processing time needed for repositioning when it is not necessary. For example, executing the APPEND procedure with the DDBLOAD = 1 would decrease processing time.

Do not set DDBLOAD equal to 1 for SAS procedures such as FSEDIT that reposition on a newly added data record.

---

## DDBLOCK= Data Set Option

Suppresses the creation of an interface spool file.

Default value: 0

---

### Syntax

DDBLOCK= 0 | 1

**0**

causes the interface view engine to create an intermediate file (spool file), which holds the data to be processed by the SAS program.

**1**

suppresses creation of the spool file, assuming that only one user is accessing the database table or that the database table is locked by some method separate from SAS.

### Details

SAS and CA-Datcom/DB use different locking mechanisms. The interface view engine compensates by creating a spool file of static data, which can adversely affect performance. The DDBLOCK= data set option suppresses creation of the spool file and lets static data be provided another way.

---

## DDBPW= Data Set Option

Specifies a CA-DATADictionary password.

---

### Syntax

DDBPW=*'password'*

**'password'**

is the 12-character PASSWORD attribute of the PERSON entity-occurrence for the specified userid.

**Details**

The DDBPW= data set option specifies an optional CA-DATADictionary password. If one is specified in the view descriptor, this data set option overrides it. If CA-DATADictionary requires a password and the view descriptor does not include one, you must specify the password with the DDBPW= data set option.

Not every userid requires a password.

**DDBSV= Data Set Option**

Specifies the status and version of the CA-Datcom/DB table that you want to access.

**Syntax**

**DDBSV=**'*status / version*'

**'status/version'**

is a 4-character field. A status is either PROD, TEST, or T (for TEST) plus a 3-digit number. The default is PROD.

**Details**

The DDBSV= data set option specifies an optional CA-DATADictionary status value. If one is specified in the view descriptor, this data set option overrides it. Other status values, such as HIST, are not used. Status/version can be changed any time.

**DDBTRACE= Data Set Option**

Displays information for debugging purposes.

Default value: 0

**Syntax**

**DDBTRACE=** 0 | 1

- 0**  
does not display trace information.
- 1**  
displays trace information.

### Details

The DDBTRACE= data set option can be used to produce traces for debugging purposes. Contact Technical Support at SAS if you need more information. It also enables you to display the WHERE clause that is passed to CA-Datacom/DB, and a message indicating whether a temporary index will be created.

---

## DDBURT= Data Set Option

Specifies the URT that will be used by the interface view engine when it opens the view descriptor.

---

### Syntax

**DDBURT**=*'User-Requirements-Table-name'*

### Details

The interface view engine will create a default URT unless you stored one as a system option, included one in the view descriptor, or specified one with the DDBURT= data set option in the SAS program.

---

## DDBUSER= Data Set Option

Specifies the CA-DATADictionary userid.

---

### Syntax

**DDBUSER**=*'userid'*

#### *'userid'*

is the 32-character entity-occurrence name of a PERSON entity in the CA-DATADictionary database.

### Details

The userid is required. If you specify this data set option, it overrides the userid in the view descriptor.

The CA-Datcom/DB userid is not necessarily the same as the user's TSO ID. The userid and optional password must have retrieval authority on six entity-types: DATABASE, FILE, RECORD, KEY, ELEMENT, and FIELD.

---

## Using Multiple View Descriptors

The SAS/ACCESS interface to CA-Datcom/DB supports having multiple views (view descriptors) open simultaneously. Multiple views can be open for updating as well as retrievals.

---

## User Exits from CA-Datcom/DB

Two user exits are available to you. You can set your own default SAS names with a user exit in the ACCESS procedure. You can also do security checking with a user exit that is available in the interface view engine. For more information about user exits, contact your on-site SAS support personnel.

---

## Deleting and Inserting Data Records in CA-Datcom/DB

Deleting and inserting data records in a CA-Datcom/DB table is a straightforward process for SAS as well as for CA-Datcom/DB. However, several considerations are worth noting.

- When inserting a new row in SAS, you normally visualize the new data being appended at the end of the SAS data file. However, CA-Datcom/DB conventions affect the appearance of a procedure such as FSEDIT.

When you include a WHERE clause or criteria that require a Select file, the customary SAS behavior applies. Otherwise, newly added data records appear in order by the Native Key, not at the end. The new records are always inserted in the table in the proper place, in order by the Native Key. Including or not including a WHERE clause simply determines how the table looks to SAS.

*Note:* If you are going to use a view descriptor to insert new data records, you must select at least one key (or the fields that make up the key) for that view descriptor. Select the Master Key if the Master Key values must be unique. If you request repositioning (that is, the DDBLOAD= option is set to zero or default to zero), either the default key must be selected or all of the fields that it comprises must be selected. An ADD cannot be processed if DDBLOAD equals 0 and the Default Key field is set to blanks.  $\triangle$

- If INCLUDE-NIL-KEY equals YES, null values are indexed. However, if DUPE-MASTER-KEY equals NO, duplicate values cannot be indexed. That is, if you insert two records with missing values for the Master Key, the second insert fails. Native Key values must be unique, another reason that the second insert of a missing value fails.
- In order for you to even see newly added records while in an interactive procedure, the interface view engine does some fairly complex processing that is referred to as repositioning. Repositioning means retrieving a record immediately after it has been inserted, so that it is visible to you. This is important in an interactive procedure, but not as important in a procedure such as PROC APPEND.

Therefore, to save processing time, a DDBLOAD= data set option is available that enables you to turn repositioning on and off. Setting the DDBLOAD= data set option causes the APPEND procedure, for example, to process faster and more efficiently. For more information about the DDBLOAD= data set option, see “Data Set Options” on page 117.

- Deleting a data record removes the record from the CA-Datcom/DB table. The deleted data record is no longer available for access even though it might have satisfied the WHERE clause selection criteria before it was removed.

---

## Missing Values (Nils) in CA-Datcom/DB Tables

When the interface view engine stores a missing value in a CA-Datcom/DB table, the missing value is represented by all blanks for both numeric and character type fields. When the engine retrieves a missing value, it appears as all blanks for SAS character columns and as a period (.) for SAS numeric columns.

*Note:* A system option is available if you want to set missing values to zeros.  $\Delta$

---

## SAS WHERE Clause Conditions Not Acceptable to CA-Datcom/DB

Here is a list of some (but not all) SAS WHERE clause conditions that are not acceptable to CA-Datcom/DB; they are handled automatically by SAS post-processing.

- arithmetic expressions, for example,

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

- expressions in which a column or combination of columns assumes a value of 1 or 0 to signify true or false, for example,

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

- concatenation of character columns
- truncated comparison, for example,

```
c1=:abc
```

- DATETIME and TIME formats, for example,

```
'12:00't
'01jan60:12:00'dt
```

- SOUNDEX
- HAVING, GROUP BY
- references to missing values. This includes the period (.) for numeric columns, blanks for character columns, and the IS MISSING and IS NULL operators.

---

## Deciding How to Specify Selection Criteria in CA-Datcom/DB

---

---

### WHERE Clause in the View Descriptor

Include a WHERE clause in your view descriptor when you want to do the following tasks:

- restrict users of view descriptors to certain subsets of data
- use CA-Datcom/DB syntax and functionality, such as masking or delimiters
- prevent users from sequentially passing the entire CA-Datcom/DB table.

---

### SAS WHERE Clause

Use a SAS WHERE clause when the previous guidelines do not apply and you want to do the following tasks:

- have more run-time flexibility in subsetting data
- use SAS WHERE clause capabilities that CA-Datcom/DB does not support, such as arithmetic expressions or truncated comparisons.

*Note:* Masking and delimiter characters are ignored in a SAS WHERE clause. For example,

```
where x='\y\a'
```

would cause the software to look for backslashes in the data.  $\triangle$

---

## Validation of Data Values in CA-Datcom/DB

The interface view engine does not validate the incoming data. That is, it does not check to see whether invalid data was entered in a CA-Datcom/DB table, for example, nonpacked data into a packed type field. However, it does check for SAS format return codes from retrievals and moves missing values into the retrieved row if errors occur. No checking takes place for updates or WHERE clause processing.

---

## Validation against CA-DATADICIONARY

The interface view engine always reads in the CA-DATADICIONARY information when it processes a view descriptor, because CA-DATADICIONARY contains information that is not in the view descriptor, for example, the element entity.

Consider the following matters regarding CA-DATADICIONARY usage by the interface view engine:

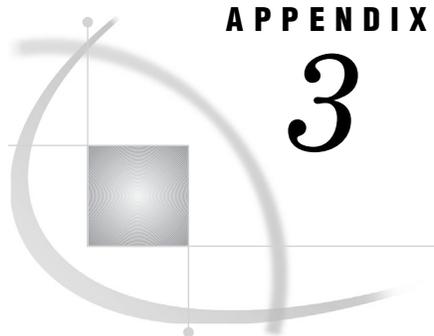
- Elements are groupings of fields. They are used to read and update the tables. The interface view engine makes a "best-fit" list of elements to use when it executes the view descriptor. If the database administrator fine-tunes the element definitions to improve performance, the interface view engine can take advantage of the improvements.

- The interface view engine rejects a view descriptor if certain information that is stored in the view descriptor does not match the dictionary. Specifically, you cannot change the following information:
  - entity-occurrence names and types for these entities: DATABASE, FILE, RECORD, KEY, FIELD
  - field class, from simple to compound or vice versa
  - field data types, from one category to another. The categories are as follows:
    - numeric*: types B (binary), 2 (halfword), 4 (fullword), 8 (double), N(Zoned), D (packed decimal)
    - float*: types L (long float), S (short float)
    - alpha*: types C (char), H (hex)
    - Xtype*: types B > 8 bytes, D and N > 16 bytes, E (extended float), G (graphics), K (kanji), T (PL/1 bit), Y (double byte character set), Z (mixed DBCS and single byte)

*Note:* Xtype is an artificial type. It represents a CA-Datacom/DB type that cannot be fully handled in SAS. △

- You should set up a CA-DATADictionary user ID and password for SAS and not change them. The user ID should have retrieval authority in CA-DATADictionary for six entity-types: DATABASE, FILE, RECORD, ELEMENT, KEY, and FIELD. The user ID and password do not protect data, but the interface view engine needs them to obtain information from CA-DATADictionary. If you change the user ID or password, then the view descriptors must be recreated, or you must use the data set options to override the user ID and password. (You can also zap these values in the CSECT DDBAUSE option, which is used by the ACCESS procedure.)





## APPENDIX

## 3

## Data and Descriptors for the Examples

---

<i>Introduction to Data and Descriptors for the Examples</i>	129
<i>CA-Datcom/DB Tables</i>	130
<i>Using the CA-Datcom/DB Tables</i>	130
<i>CA-DATADictionary Statements for Sample Tables</i>	130
<i>CUSTOMERS Table</i>	136
<i>EMPLOYEES Table</i>	139
<i>INVOICE Table</i>	141
<i>ORDER Table</i>	142
<i>Access Descriptors for the CA-Datcom/DB Tables</i>	144
<i>MYLIB.CUSTS Access Descriptor</i>	144
<i>MYLIB.EMPLOYEE Access Descriptor</i>	145
<i>MYLIB.INVOICE Access Descriptor</i>	145
<i>MYLIB.ORDERS Access Descriptor</i>	145
<i>View Descriptors for the CA-Datcom/DB Tables</i>	146
<i>VLIB.ALLEMP View Descriptor</i>	146
<i>VLIB.ALLORDR View Descriptor</i>	146
<i>VLIB.CUSORDR View Descriptor</i>	146
<i>VLIB.CUSPHON View Descriptor</i>	146
<i>VLIB.CUSTADD View Descriptor</i>	147
<i>VLIB.DCMEMPS View Descriptor</i>	147
<i>VLIB.EMPINFO View Descriptor</i>	147
<i>VLIB.EMPS View Descriptor</i>	147
<i>VLIB.FORINV View Descriptor</i>	147
<i>VLIB.INV View Descriptor</i>	148
<i>VLIB.USACUST View Descriptor</i>	148
<i>VLIB.USAINV View Descriptor</i>	148
<i>VLIB.USAORDR View Descriptor</i>	149
<i>SAS Data Files Used for CA-Datcom/DB Examples</i>	149
<i>MYDATA.OUTOFSTK Data File</i>	149
<i>MYDATA.SASEMPS Data File</i>	150
<i>LIB6.BIRTHDAY Data File</i>	150

---

### Introduction to Data and Descriptors for the Examples

This appendix gives information about the CA-Datcom/DB tables, descriptor files, and SAS data files used in the examples in this document. It shows the CA-DATADictionary statements and the data that were used to build the CA-Datcom/DB tables. It also displays the access descriptors and view descriptors, along with selection criteria specified for them. 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, contact your on-site SAS support personnel for information.

---

## CA-Datcom/DB Tables

---

### Using the CA-Datcom/DB Tables

This section describes the CA-Datcom/DB tables referenced in this document. It shows the following information:

- the CA-DATADictionary statements and data for the tables
- the Native Key and the Master Key for each table
- the compound fields and repeating fields, if any

The four CA-Datcom/DB tables used in the examples are named CUSTOMERS, EMPLOYEES, INVOICE, and ORDER. They are all in one CA-Datcom/DB database named TEXTILES. To build the tables, follow these steps:

- 1 Create the CA-DATADictionary entries and catalog them to CA-Datcom/DB.
- 2 Create the SAS data files.
- 3 Create an access descriptor and an associated view descriptor for each table. Make sure that all SAS names in the view descriptors match the names in the SAS data files. Use the access descriptors in this appendix as a model. Select every field for the access descriptors, and create view descriptors that also select every field.
- 4 Run the APPEND procedure with the data set options shown here:

```
proc append data=SAS-file base=view-descriptor; run;
```

---

### CA-DATADictionary Statements for Sample Tables

Here are the CA-DATADictionary statements used to create the four sample CA-Datcom/DB tables. This is input to the DDUPDATE utility.

```
//SYSIN DD *
-USR ADR-INSTALL,NEWUSER
-ADD DATABASE,TEXTILES(T001)
3000 030
-END
-ADD AREA,TEX030(T001)
1000 CONNECT,TEXTILES
3001 TEX SASBXB.Datcom.TEX030
3002     TEX030           3380   04096
-END
-ADD FILE,CUSTOMF(T001)
1000 CONNECT,TEX030
3100 DB           01024           FBLK Y
3101 CUS 001 Y Y           ADR/DB
-END
-ADD RECORD,CUSTOMERS(T001)
1000 CONNECT,CUSTOMF
3200 DB
```

```

-END
-GRP START,RECORD,CUSTOMERS(T001)
-ADD FIELD,CUSTOMER
4010 START START
4012 S C L N 00008 00 00001
-END
-ADD FIELD,STATEZIP
4010 CUSTOMER START
4012 C C L N 00007 00 00001
-END
-ADD FIELD,STATE
4010 STATEZIP STATEZIP
4012 S C L N 00002 00 00001
-END
-ADD FIELD,ZIPCODE
4010 STATE STATEZIP
4012 S N R N 00005 00 00001
-END
-ADD FIELD,COUNTRY
4010 STATE START
4012 S C L N 00020 00 00001
-END
-ADD FIELD,TELEPHONE
4010 COUNTRY START
4012 S C L N 00012 00 00001
-END
-ADD FIELD,NAME
4010 TELEPHONE START
4012 S C L N 00060 00 00001
-END
-ADD FIELD,CONTACT
4010 NAME START
4012 S C L N 00030 00 00001
-END
-ADD FIELD,STREETADDRESS
4010 CONTACT START
4012 S C L N 00040 00 00001
-END
-ADD FIELD,CITY
4010 STREETADDRESS START
4012 S C L N 00025 00 00001
-END
-ADD FIELD,FIRSTORDERDATE
4010 CITY START
4012 S N R N 00006 00 00001
-END
-ADD FIELD,SIGNATURELIST
4010 FIRSTORDERDATE START
4012 C C L N 00044 00 00005
-END
-ADD FIELD,LIMIT
4010 SIGNATURELIST SIGNATURELIST
4012 S N R N 00014 02 00001
-END

```

```

-ADD FIELD,SIGNATURE
4010 LIMIT SIGNATURELIST
4012 S C L N 00030 00 00001
-END
-ADD FIELD,BRANCHOFFICE
4010 SIGNATURE START
4012 S C L N 00025 00 00010
-END
-GRP END
-ADD KEY,CUSTOMERS.CUSKEY(T001)
5000 CUSKY 001 Y Y Y
5010 ADD CUSTOMER
5011 $FIRST
-END
-ADD ELEMENT,CUSTOMERS.CUSELM(T001)
6000 CUSEL
6010 ADD CUSTOMER
6010 ADD STATE
6010 ADD ZIPCODE
6010 ADD COUNTRY
6010 ADD TELEPHONE
6010 ADD NAME
6010 ADD CONTACT
6010 ADD STREETADDRESS
6010 ADD CITY
6010 ADD FIRSTORDERDATE
6010 ADD SIGNATURELIST
6010 ADD BRANCHOFFICE
-END
-ADD FILE,EMPLOYF(T001)
1000 CONNECT,TEX030
3100 DB 01024 FBLK Y
3101 EMP 002 Y Y ADR/DB
-END
-ADD RECORD,EMPLOYEES(T001)
1000 CONNECT,EMPLOYF
3200 DB
-END
-GRP START,RECORD,EMPLOYEES(T001)
-ADD FIELD,EMPID
4010 START START
4012 S N R N 00006 00 00001
-END
-ADD FIELD,HIREDATE
4010 EMPID START
4012 S N R N 00006 00 00001
-END
-ADD FIELD,SALARY
4010 HIREDATE START
4012 S N R N 00007 02 00001
-END
-ADD FIELD,DEPT
4010 SALARY START
4012 S C L N 00006 00 00001

```

```

-END
-ADD FIELD, JOBCODE
4010 DEPT START
4012 S N R N 00005 00 00001
-END
-ADD FIELD, SEX
4010 JOBCODE START
4012 S C L N 00001 00 00001
-END
-ADD FIELD, BIRTHDATE
4010 SEX START
4012 S N R N 00006 00 00001
-END
-ADD FIELD, LASTNAME
4010 BIRTHDATE START
4012 S C L N 00018 00 00001
-END
-ADD FIELD, FIRSTNAME
4010 LASTNAME START
4012 S C L N 00015 00 00001
-END
-ADD FIELD, MIDDLENAME
4010 FIRSTNAME START
4012 S C L N 00015 00 00001
-END
-ADD FIELD, PHONE
4010 MIDDLENAME START
4012 S C L N 00004 00 00001
-END
-GRP END
-ADD KEY, EMPLOYEES.EMPKEY(T001)
5000 EMPKY 002 Y Y Y
5010 ADD EMPID
5011 $FIRST
-END
-ADD ELEMENT, EMPLOYEES.EMPELM(T001)
6000 EMPEL
6010 ADD EMPID
6010 ADD HIREDATE
6010 ADD SALARY
6010 ADD DEPT
6010 ADD JOBCODE
6010 ADD SEX
6010 ADD BIRTHDATE
6010 ADD LASTNAME
6010 ADD FIRSTNAME
6010 ADD MIDDLENAME
6010 ADD PHONE
-ADD FILE, INVOICF(T001)
1000 CONNECT, TEX030
3100 DB 01024 FBLK Y
3101 INV 003 Y Y ADR/DB
-END
-ADD RECORD, INVOICE(T001)

```

```

1000 CONNECT, INVOICF
3200 DB
-END
-GRP START, RECORD, INVOICE(T001)
-ADD FIELD, INVOICENUM
4010 START START
4012 S N R N 00005 00 00001
-END
-ADD FIELD, BILLEDTO
4010 INVOICENUM START
4012 S C L N 00008 00 00001
-END
-ADD FIELD, AMTBILLED
4010 BILLEDTO START
4012 S N R N 00014 02 00001
-END
-ADD FIELD, COUNTRY
4010 AMTBILLED START
4012 S C L N 00020 00 00001
-END
-ADD FIELD, AMOUNTINUS
4010 COUNTRY START
4012 S N R N 00010 02 00001
-END
-ADD FIELD, BILLEDBY
4010 AMOUNTINUS START
4012 S N R N 00006 00 00001
-END
-ADD FIELD, BILLEDON
4010 BILLEDBY START
4012 S N R N 00006 00 00001
-END
-ADD FIELD, PAIDON
4010 BILLEDON START
4012 S N R N 00006 00 00001
-END
-ADD FIELD, COMPUTEDEXCHANG
4010 PAIDON START
4012 S L R Y 00008 00 00001
-END
-GRP END
-ADD KEY, INVOICE.INVKEY(T001)
5000 INVKY 003 Y Y Y
5010 ADD INVOICENUM
5011 $FIRST
-END
-ADD ELEMENT, INVOICE.INVELM(T001)
6000 INVEL
6010 ADD INVOICENUM
6010 ADD BILLEDTO
6010 ADD AMTBILLED
6010 ADD COUNTRY
6010 ADD AMOUNTINUS
6010 ADD BILLEDBY

```

```

6010 ADD BILLEDON
6010 ADD PAIDON
6010 ADD COMPUTEDEXCHANG
-ADD FILE,ORDERF(T001)
1000 CONNECT,TEX030
3100 DB          01024          FBLK Y
3101 ORD 004 Y Y          ADR/DB
-END
-ADD RECORD,ORDER(T001)
1000 CONNECT,ORDERF
3200 DB
-END
-GRP START,RECORD,ORDER(T001)
-ADD FIELD,ORDERNUM
4010 START
4012 S N R N 00005 00 00001
-END
-ADD FIELD,STOCKNUM
4010 ORDERNUM
4012 S N R N 00004 00 00001
-END
-ADD FIELD,LENGTH
4010 STOCKNUM
4012 S N R N 00004 00 00001
-END
-ADD FIELD,FABRICCHARGES
4010 LENGTH
4012 S N R N 00010 02 00001
-END
-ADD FIELD,SHIPTO
4010 FABRICCHARGES
4012 S C L N 00008 00 00001
-END
-ADD FIELD,DATEORDERED
4010 SHIPTO
4012 S N R N 00006 00 00001
-END
-ADD FIELD,SHIPPED
4010 DATEORDERED
4012 S N R N 00006 00 00001
-END
-ADD FIELD,TAKENBY
4010 SHIPPED
4012 S N R N 00006 00 00001
-END
-ADD FIELD,PROCESSEDBY
4010 TAKENBY
4012 S N R N 00006 00 00001
-END
-ADD FIELD,SPECIALINSTRUCT
4010 PROCESSEDBY
4012 S C L N 00001 00 00001
-END
-GRP END

```

```

-ADD KEY,ORDER.ORDKEY(T001)
5000 ORDKY 004 Y Y          Y
5010 ADD ORDERNUM
5011 $FIRST
-END
-ADD ELEMENT,ORDER.ORDELM(T001)
6000 ORDEL
6010 ADD ORDERNUM
6010 ADD STOCKNUM
6010 ADD LENGTH
6010 ADD FABRICCHARGES
6010 ADD SHIPTO
6010 ADD DATEORDERED
6010 ADD SHIPPED
6010 ADD TAKENBY
6010 ADD PROCESSEDBY
6010 ADD SPECIALINSTRUCT
//

```

---

## CUSTOMERS Table

The sample CA-Datcom/DB table named CUSTOMERS is in the TEXTILES database. The user ID is DEMO. CUSKEY is the Native Key and the Master Key. CUSKEY consists of the CUSTOMER field, which contains a unique integer number for each customer.

In the CUSTOMERS table, STATEZIP is a compound field, consisting of the two simple fields STATE and ZIPCODE at level 2. SIGNATURELIST is also a compound field; it consists of the LIMIT field and the SIGNATURE field. The data values for this compound field indicate whose signature is required for specific amounts of money. This compound field is also a repeating field. The last field, BRANCHOFFICE, is a repeating field that can occur ten times. It is a simple field containing city names of branch offices for the customer.

Here are the fields in the CUSTOMERS table. The data is shown in Output A3.1.

```

DATA CUSTOMER;
      INPUT @1  CUSTNUM      $8.          /* CUSTOMER NUMBER          */
           @10 STATE        $2.
           @13 ZIPCODE      5.          /* ZIPCODE IF COMPANY IS   */
                                           /* IN THE U.S., OTHERWISE */
                                           /* IT IS THE MAIL CODE     */
                                           /* APPROPRIATE FOR THE     */
                                           /* COUNTRY WHERE THE       */
                                           /* COMPANY IS LOCATED      */
           @20 COUNTRY      $20.
           @42 PHONE        $12. /
           @1  NAME         $60. / /* CUSTOMER'S COMPANY NAME*/
           @1  CONTACT      $30. /* CONTACT AT CUSTOMER'S  */
                                           /* COMPANY                  */
           @31 STREET       $40. /
           @1  CITY         $25.
           @30 FIRSTORD     YYMMDD6./ /* DATE OF FIRST ORDER     */
           @1  LIMIT1       15.2 /* SIGNATURE LIMIT #1     */
           @20 SIGNATU1     $30. / /* SIGNATURE NAME #1     */
           @1  LIMIT2       15.2 /* SIGNATURE LIMIT #2     */

```

```

@20 SIGNATU2 $30. / /* SIGNATURE NAME #2 */
@1 LIMIT3 15.2 /* SIGNATURE LIMIT #3 */
@20 SIGNATU3 $30. / /* SIGNATURE NAME #3 */
@1 BRANCHO1 $25. /* BRANCH OFFICE #1 */
@30 BRANCHO2 $25. / /* BRANCH OFFICE #2 */
@1 BRANCHO3 $25. /* BRANCH OFFICE #3 */
@30 BRANCHO4 $25.; /* BRANCH OFFICE #4 */

```

```
FORMAT FIRSTORD DATE7.;
```

### Output A3.1 Data for CUSTOMERS Table

CUSTOMER	NAME	COUNTRY	TELEPHON
12345678			919/489-5682
14324742	SANTA CLARA VALLEY TECHNOLOGY SPECIALISTS	USA	408/629-0589
14569877	PRECISION PRODUCTS	USA	919/489-6792
14898029	UNIVERSITY BIOMEDICAL MATERIALS	USA	301/760-2541
15432147	GREAT LAKES LABORATORY EQUIPMENT MANUFACTURERS	USA	616/582-3906
18543489	LONE STAR STATE RESEARCH SUPPLIERS	USA	512/478-0788
19783482	TWENTY-FIRST CENTURY MATERIALS	USA	703/714-2900
19876078	SAN JOAQUIN SCIENTIFIC AND INDUSTRIAL SUPPLY, INC.	USA	209/686-3953
24589689	CENTAR ZA TECHNICKU I NAUCNU RESTAURIRANJE UMJETNINA	Yugoslavia	(012)736-202
26422096	SOCIETE DE RECHERCHES POUR DE CHIRURGIE ORTHOPEDIQUE	France	4268-54-72
26984578	INSTITUT FUR TEXTIL-FORSCHUNGS	Austria	43-57-04
27654351	INSTITUT DE RECHERCHE SCIENTIFIQUE MEDICALE	Belgium	02/215-37-32
28710427	ANTONIE VAN LEEUWENHOEK VERENIGING VOOR MICROBIOLOGIE	Netherlands	(021)570517
29834248	BRITISH MEDICAL RESEARCH AND SURGICAL SUPPLY	Britain	(0552)715311
31548901	NATIONAL COUNCIL FOR MATERIALS RESEARCH	Canada	406/422-3413
38763919	INSTITUTO DE BIOLOGIA Y MEDICINA NUCLEAR	Argentina	244-6324
39045213	LABORATORIO DE PESQUISAS VETERINARIAS DESIDERIO FINAMOR	Brazil	012/302-1021
43290587	HASSEI SAIBO GAKKAI	Japan	(02)933-3212
43459747	RESEARCH OUTFITTERS	Australia	03/734-5111
46543295	WESTERN TECHNOLOGICAL SUPPLY	Japan	(03)022-2332
46783280	NGEE TECHNOLOGICAL INSTITUTE	Singapore	3762855
48345514	GULF SCIENTIFIC SUPPLIES	United Arab Emirates	213445

CUSTOMER	STREETAD	CITY	STATE	ZIPCODE
12345678			NC	.
14324742	5089 CALERO AVENUE	SAN JOSE	CA	95123
14569877	198 FAYETTVILLE ROAD	MEMPHIS	NC	27514
14898029	1598 PICCARD DRIVE	ROCKVILLE	MD	20850
15432147	103 HARRIET STREET	KALAMAZOO	MI	49001
18543489	5609 RIO GRANDE	AUSTIN	TX	78701
19783482	4613 MICHAEL FARADAY DRIVE	RESTON	VA	22090
19876078	1095 HIGHWAY 99 SOUTH	TULARE	CA	93274
24589689	TAKOVSKA 4	BELGRADE		.
26422096	40 RUE PERIGNON	LA ROCHELLE		75014
26984578	MECHITARISTENGASSE 5	VIENNA		5110
27654351	103 RUE D'EGMONT	BRUSSELS		5010
28710427	BIRMOERSTRAAT 34	THE HAGUE	HV	3607
29834248	44 PRINCESS GATE, HYDE PARK	LONDON, SW7 1PU		.
31548901	5063 RICHMOND MALL	VANCOUVER, V5T 1L2	BC	.
38763919	SALGUERO 2345	BUENOS AIRES		1405
39045213	RUA DONA ANTONIA DE QUEIROS 381	SAO PAULO	SP	1051
43290587	3-2-7 ETCHUJMA, KOTO-KU	TOKYO 101		.
43459747	191 LOWER PLENTY ROAD	PRAHRAN, VICTORIA		3181
46543295	4-3-8 ETCHUJMA, KOTO-KU	TOKYO 102		.
46783280	356 CLEMENTI ROAD	SINGAPORE		2374
48345514	POB 8032	RAS AL KHAIMAH		.

CUSTOMER	CONTACT	FIRSTORD	LIMIT	SIGNATU
12345678		.	.	
14324742	A. BAUM	05FEB65	5000.00	BOB HENSON
			25000.00	KAREN DRESSER
14569877	CHARLES BARON	15AUG83	5000.00	JEAN CRANDALL
			100000.00	STEVE BLUNTSEN
14898029	S. TURNER	12NOV76	10000.00	MASON FOXWORTH
			50000.00	DANIEL STEVENS
			100000.00	ELIZABETH PATTON
15432147	D.W. KADARAUCH	28APR86	10000.00	JACK TREVANE
18543489	A. SILVERIA	10SEP79	10000.00	NANCY WALSH
			50000.00	TED WHISTLER
			100000.00	EVAN MASSEY
19783482	M.R. HEFFERNAN	18JUL68	5000.00	PETER THOMAS
			10000.00	LOUIS PICKERING
19876078	J.A. WHITTEN	11MAY79	7500.00	EDWARD LOWE
			25000.00	E.F. JENSEN
24589689	J.V. VUKASINOVIC	30NOV81	.	
26422096	Y. CHAVANON	14JUN83	5000.00	MICHELE PICARD
			10000.00	M.L. SEIS
26984578	GUNTER SPIELMANN	25MAY87	100000.00	FRANZ BECH
27654351	I. CLEMENS	14OCT86	5000.00	C.J. HELMER
28710427	M.C. BORGSTEEDE	10OCT85	10000.00	J.J. JASPER
29834248	A.D.M. BRYCESON	29JAN86	5000.00	ELVIN POMEROY
31548901	W.E. MACDONALD	19MAR84	1000.00	DAPHNE MARSHALL
38763919	JORGE RUNNAZZO	10DEC84	2500.00	M.L. CARLOS
39045213	ELISABETE REGIS GUILLAUMON	18AUG82	1500.00	RICK ESTABAN
43290587	Y. FUKUDA	08FEB74	10000.00	R. YAMOTO
43459747	R.G. HUGHES	28JUL72	1000.00	DENNIS RICHMOND
			5000.00	JANICE HEATH
46543295		19APR84	10000.00	DAPHNE MARSHALL
46783280	LING TAO SOON	27SEP79	.	
48345514	J.Q. RIFAI	10SEP86	.	

CUSTOMER	BRANCHO1	BRANCHO2	BRANCHO3	BRANCHO4
12345678				
14324742	TORONTO	HOUSTON	TOKYO	LONDON
14569877	NEW YORK	CHICAGO	LOS ANGELES	
14898029	NEW YORK	CHICAGO	DALLAS	
15432147	CHICAGO	COLUMBUS		
18543489	HOUSTON	DALLAS	EL PASO	LUBBOCK
19783482	WASHINGTON D.C.	NEW YORK		
19876078				
24589689				
26422096	LONDON	NEW YORK		
26984578	LONDON	NEW YORK	ROME	
27654351	LONDON	BOSTON		
28710427	LONDON			
29834248	SINGAPORE	TORONTO	CAIRO	
31548901	SEATTLE	TORONTO		
38763919	MIAMI	NEW YORK		
39045213	MIAMI	NEW YORK		
43290587	SAN FRANCISCO			
43459747	SEATTLE			
46543295	SEATTLE	TORONTO	SAN FRANCISCO	DENVER
46783280				
48345514				

---

## EMPLOYEES Table

The sample CA-Datcom/DB table named EMPLOYEES is in the TEXTILES database. The user ID is DEMO. EMPKEY is the Native Key and the Master Key. It consists of the numeric field EMPID, which contains a unique integer number for each employee.

Here are the fields in the EMPLOYEE table. The data is shown in Output A3.2.

```
DATA EMPLOY;
  INPUT @1  EMPID      6.           /* EMPLOYEE ID NUMBER      */
        @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.;
```

## Output A3.2 Data for EMPLOYEES Table

OBS	EMPID	HIREDATE	SALARY	DEPT	JOBCODE	SEX	BIRTHDAT	LASTNAME
1	119012	01JUL68	42340.58	CSR010	602	F	05JAN46	WOLF-PROVENZA
2	120591	05DEC80	31000.55	SHP002	602	F	12FEB46	HAMMERSTEIN
3	123456	04APR89	.	.	.	.	.	VARGAS
4	127845	16JAN67	75320.34	ACC024	204	M	25DEC43	MEDER
5	129540	01AUG82	56123.34	SHP002	204	F	31JUL60	CHOULAI
6	135673	15JUL84	46322.58	ACC013	602	F	21MAR61	HEMESLY
7	212916	15FEB51	52345.58	CSR010	602	F	29MAY28	WACHBERGER
8	216382	15JUN85	34004.65	SHP013	602	F	24JUL63	PURINTON
9	234967	19DEC88	17000.00	CSR004	602	M	21DEC67	SMITH
10	237642	01NOV76	43200.34	SHP013	602	M	13MAR54	BATTERSBY
11	239185	07MAY81	57920.66	ACC024	602	M	28AUG59	DOS REMEDIOS
12	254896	04APR85	35000.74	CSR011	204	M	06APR49	TAYLOR-HUNYADI
13	321783	10SEP67	48931.58	CSR011	602	M	03JUN35	GONZALES
14	328140	10JAN75	75000.34	ACC043	1204	F	02JUN51	MEDINA-SIDONIA
15	346917	02MAR87	46000.33	SHP013	204	F	15MAR50	SHIEKELESLAM
16	356134	14JUN85	62450.75	ACC013	204	F	25OCT60	DUNNETT
17	423286	19DEC88	32870.66	ACC024	602	M	31OCT64	MIFUNE
18	456910	14JUN78	45000.58	CSR010	602	M	24SEP53	ARDIS
19	456921	19AUG87	33210.04	SHP002	602	M	12MAY62	KRAUSE
20	457232	15JUL85	55000.66	ACC013	602	M	15OCT63	LOVELL
21	459287	02NOV64	50000.00	SHP024	204	M	05JAN34	RODRIGUES
22	677890	12DEC88	37610.00	CSR010	204	F	24APR65	NISHIMATSU-LYNCH

OBS	FIRSTNAM	MIDDLENA	PHONE
1	G.	ANDREA	3467
2	S.	RACHAEL	3287
3	PAUL	JESUS	
4	VLADIMIR	JORAN	6231
5	CLARA	JANE	3921
6	STEPHANIE	J.	6329
7	MARIE-LOUISE	TERESA	8562
8	PRUDENCE	VALENTINE	3852
9	GILBERT	IRVINE	7274
10	R.	STEPHEN	8342
11	LEONARD	WESLEY	4892
12	ITO	MISHIMA	0231
13	GUILLEMO	RICARDO	3642
14	MARGARET	ROSE	5901
15	SHALA	Y.	8745
16	CHRISTINE	MARIE	4213
17	YUKIO	TOSHIRO	3278
18	RICHARD	BINGHAM	4351
19	KARL-HEINZ	G.	7452
20	WILLIAM	SINCLAIR	6321
21	JUAN	M.	5879
22	CAROL	ANNE	6245

---

## INVOICE Table

The sample CA-Datacom/DB table named INVOICE is in the TEXTILES database. The user ID is DEMO. INVKEY is the Native Key and the Master Key. It consists of the numeric field INVOICE, which contains a unique integer number for each invoice.

Here are the fields for the INVOICE table. The data is shown in Output A3.3.

```

DATA INVOICE;
  INPUT @1 INVOICEN 5.          /* INVOICE NUMBER          */
        @7 BILLEDTO $8.        /* COMPANY THAT PLACED THE /*
                                   /* THE ORDER                */

        @15 AMTBILLE 15.2      /* AMOUNT OF BILL IN LOCAL /*
                                   /* CURRENCY                  */

        @30 COUNTRY $20.
        @50 AMOUNTIN 11.2 /     /* AMOUNT OF BILL IN U.S.  /*
                                   /* DOLLARS                   */

        @1 BILLEDBY 6.         /* EMPLOYEE WHO WROTE THE  /*
                                   /* BILL                      */

        @10 BILLEDON YYMMDD6.  /* DATE THAT BILL WAS SENT /*

        @20 PAIDON YYMMDD6.    /* DATE THAT BILL WAS PAID /*

        @30 COMPUTED TIME8. ;  /* TIME OF DAY THAT THE    /*
                                   /* EXCHANGE RATE TO U.S.   /*
                                   /* DOLLARS WAS COMPUTED    */

  FORMAT BILLEDON DATE7.;
  FORMAT PAIDON DATE7.;

```

## Output A3.3 Data for INVOICE Table

OBS	INVOICEN	BILLEDTO	AMTBILLE	COUNTRY	AMOUNTIN
1	11270	39045213	1340738760.90	Brazil	2256870.00
2	11271	18543489	11063836.00	USA	11063836.00
3	11273	19783482	252148.50	USA	252148.50
4	11276	14324742	1934460.00	USA	1934460.00
5	11278	14898029	1400825.00	USA	1400825.00
6	11280	39045213	1340738760.90	Brazil	2256870.00
7	11282	19783482	252148.50	USA	252148.50
8	11285	38763919	34891210.20	Argentina	2256870.00
9	11286	43459747	12679156.00	Australia	11063836.00
10	11287	15432147	252148.50	USA	252148.50
11	12051	39045213	1340738760.90	Brazil	2256870.00
12	12102	18543489	11063836.00	USA	11063836.00
13	12263	19783482	252148.50	USA	252148.50
14	12468	14898029	1400825.00	USA	1400825.00
15	12471	39045213	1340738760.90	Brazil	2256870.00
16	12476	38763919	34891210.20	Argentina	2256870.00
17	12478	15432147	252148.50	USA	252148.50

OBS	BILLED	BY	BILLED	ON	PAID	ON	COMPUTED
1	239185		05OCT88		18OCT88		3.9646000000000000E+04
2	457232		05OCT88		11OCT88		.
3	239185		06OCT88		11NOV88		.
4	135673		06OCT88		20OCT88		.
5	239185		06OCT88		19OCT88		.
6	423286		07OCT88		20OCT88		5.8154000000000000E+04
7	457232		07OCT88		25OCT88		.
8	239185		10OCT88		30NOV88		5.5163000000000000E+04
9	423286		10OCT88		.		3.3827000000000000E+04
10	457232		11OCT88		04NOV88		.
11	457232		02NOV88		.		3.1185000000000000E+04
12	239185		17NOV88		.		.
13	423286		05DEC88		.		.
14	135673		24DEC88		02JAN89		.
15	457232		27DEC88		.		5.0945000000000000E+04
16	135673		24DEC88		.		3.9563000000000000E+04
17	423286		24DEC88		02JAN89		.

**ORDER Table**

The sample CA-Datcom/DB table named ORDER is in the TEXTILES database. The user ID is DEMO. ORDKEY is the Native Key and the Master Key. It consists of the numeric field ORDERNUM, which contains a unique integer number for each order. Here are the fields for the ORDER table. The data is shown in Output A3.4.

```
DATA ORDERS;
  INPUT @1 ORDERNUM 5.          /* ORDER NUMBER          */
        @6 STOCKNUM 4.         /* STOCK NUMBER          */

        @10 LENGTH 4.          /* LENGTH OF MATERIAL ORDERED */

        @15 FABRICCH 11.2      /* FABRIC CHARGES        */

        @27 SHIPTO $8.         /* CUSTOMER WHOM ORDER IS TO BE */
                                /* SHIPPED TO            */

        @35 DATEORDE YYMMDD6. /* DATE OF ORDER          */
```

```

@45 SHIPPED  YMMDD6. /* DATE THAT ORDER WAS SHIPPED */

@55 TAKENBY  6.      /* EMPLOYEE WHO TOOK THE ORDER */

@62 PROCESSE 6.      /* EMPLOYEE WHO PROCESSED THE  */
                    /* THE ORDER                          */

@69 SPECIALI $1. ;   /* THIS IS A FLAG THAT SIGNALS */
                    /*THERE ARE SPECIAL INSTRUCTIONS*/
                    /* ASSOCIATED WITH THIS ORDER. */

FORMAT DATEORDE DATE7.;
FORMAT SHIPPED DATE7.;

```

**Output A3.4** Data for ORDER Table

OBS	ORDERNUM	STOCKNUM	LENGTH	FABRICCH
1	11269	9870	690	.
2	11270	1279	1750	2256870.00
3	11271	8934	110	11063836.00
4	11272	3478	1000	.
5	11273	2567	450	252148.50
6	11274	4789	1000	.
7	11275	3478	1000	.
8	11276	1279	1500	1934460.00
9	11277	8934	100	10058033.00
10	11278	2567	2500	1400825.00
11	11279	9870	650	.
12	11280	1279	1750	2256870.00
13	11281	8934	110	11063836.00
14	11282	2567	450	252148.50
15	11283	9870	690	.
16	11284	3478	1000	.
17	11285	1279	1750	2256870.00
18	11286	8934	110	11063836.00
19	11287	2567	450	252148.50
20	11288	9870	690	.
21	11969	9870	690	.
22	12051	1279	1750	2256870.00
23	12102	8934	110	11063836.00
24	12160	3478	1000	.
25	12263	2567	450	252148.50
26	12464	4789	1000	.
27	12465	3478	1000	.
28	12466	1279	1500	1934460.00
29	12467	8934	100	10058033.00
30	12468	2567	2500	1400825.00
31	12470	9870	650	.
32	12471	1279	1750	2256870.00
33	12472	8934	110	11063836.00
34	12473	2567	450	252148.50
35	12474	9870	690	.
36	12475	3478	1000	.
37	12476	1279	1750	2256870.00
38	12477	8934	110	11063836.00
39	12478	2567	450	252148.50
40	12479	9870	690	.

OBS	ORDERNUM	SHIPTO	DATEORDE	SHIPPED	TAKENBY	PROCESSE	SPECIALI
1	11269	19876078	03OCT88	.	212916	.	
2	11270	39045213	03OCT88	19OCT88	321783	237642	X
3	11271	18543489	03OCT88	13OCT88	456910	456921	
4	11272	29834248	03OCT88	.	234967	.	
5	11273	19783482	04OCT88	14NOV88	119012	216382	
6	11274	15432147	04OCT88	.	212916	.	
7	11275	29834248	04OCT88	.	234967	.	
8	11276	14324742	04OCT88	21OCT88	321783	120591	X
9	11277	31548901	05OCT88	.	456910	.	
10	11278	14898029	05OCT88	20OCT88	119012	456921	
11	11279	48345514	05OCT88	.	212916	.	
12	11280	39045213	06OCT88	21OCT88	321783	237642	X
13	11281	18543489	06OCT88	27OCT88	456910	216382	
14	11282	19783482	06OCT88	26OCT88	119012	456921	
15	11283	18543489	07OCT88	.	212916	.	
16	11284	24589689	07OCT88	.	234967	.	
17	11285	38763919	07OCT88	02DEC88	321783	120591	X
18	11286	43459747	07OCT88	03NOV88	456910	237642	
19	11287	15432147	07OCT88	07NOV88	119012	216382	
20	11288	14324742	10OCT88	.	212916	.	Y
21	11969	19876078	25OCT88	.	212916	.	
22	12051	39045213	31OCT88	.	321783	.	X
23	12102	18543489	15NOV88	.	456910	.	
24	12160	29834248	19NOV88	.	234967	.	Z
25	12263	19783482	01DEC88	.	119012	.	
26	12464	15432147	23DEC88	.	212916	.	
27	12465	29834248	23DEC88	.	234967	.	
28	12466	14324742	23DEC88	.	321783	.	X
29	12467	31548901	23DEC88	.	456910	.	
30	12468	14898029	23DEC88	03JAN89	119012	120591	
31	12470	48345514	23DEC88	.	212916	.	
32	12471	39045213	23DEC88	.	321783	.	X
33	12472	18543489	23DEC88	03JAN89	456910	237642	
34	12473	19783482	23DEC88	.	119012	.	
35	12474	18543489	23DEC88	.	212916	.	
36	12475	24589689	23DEC88	.	234967	.	
37	12476	38763919	23DEC88	03JAN89	321783	456921	X
38	12477	43459747	23DEC88	.	456910	.	
39	12478	15432147	23DEC88	03JAN89	119012	216382	
40	12479	14324742	23DEC88	.	212916	.	

## Access Descriptors for the CA-Datcom/DB Tables

### MYLIB.CUSTS Access Descriptor

The MYLIB.CUSTS access descriptor for the CUSTOMERS table was created as follows:

```
proc access dbms=Datcom;
  create mylib.custs.access;
  user=demo;
  table=customers;
  assign = yes;
  drop contact;
  list all;
  extend all;
```

```

        rename customer = custnum telephone = phone
            streetaddress = street;
        format firstorderdate = date7.;
        informat firstorderdate = date7.;
        content firstorderdate = yymmdd6.;
        list all;
run;

```

---

## MYLIB.EMPLOYEE Access Descriptor

The MYLIB.EMPLOYEE access descriptor for the EMPLOYEES table was created as follows:

```

proc access dbms=Datacom;
    create mylib.employee.access;
    user=demo;
    table=employees;
    assign = yes;
    format hiredate = date7.;
    informat hiredate = date7.;
    content hiredate = yymmdd6.;
    format birthdate = date7.;
    informat birthdate = date7.;
    content birthdate = yymmdd6.;
    list all;
    extend all;
run;

```

---

## MYLIB.INVOICE Access Descriptor

The MYLIB.INVOICE access descriptor for the INVOICE table was created as follows:

```

proc access dbms=Datacom;
    create mylib.invoice.access;
    user=demo;
    table=invoice;
    assign = yes;
    format billedon = date7.;
    informat billedon = date7.;
    content billedon = yymmdd6.;
    format paidon = date7.;
    informat paidon = date7.;
    content paidon = yymmdd6.;
    list all;
    extend all;
run;

```

---

## MYLIB.ORDERS Access Descriptor

The MYLIB.ORDERS access descriptor for the ORDER table was created as follows:

```

proc access dbms=Datacom;
    create mylib.orders.access;

```

```

user=demo;
table=order;
assign = yes;
format dateordered = date7.;
informat dateordered = date7.;
content dateordered = yymmdd6.;
format shipped = date7.;
informat shipped = date7.;
content shipped = yymmdd6.;
list all;
extend all;
run;

```

---

## View Descriptors for the CA-Datcom/DB Tables

---

### VLIB.ALLEMP View Descriptor

The VLIB.ALLEMP view descriptor was created as follows:

```

proc access dbms=Datcom ad=mylib.employee;
  create vlib.allemp.view;
  select all;
  list view;
run;

```

---

### VLIB.ALLORDR View Descriptor

The VLIB.ALLORDR view descriptor was created as follows:

```

proc access dbms=Datcom ad=mylib.orders;
  create vlib.allordr.view;
  select all;
  list view;
run;

```

---

### VLIB.CUSORDR View Descriptor

The VLIB.CUSORDR view descriptor contains no selection criteria.

```

proc access dbms=Datcom ad=mylib.orders;
  create vlib.cusordr.view;
  select stocknum shipto;
  list view;
run;

```

---

### VLIB.CUSPHON View Descriptor

The VLIB.CUSPHON view descriptor was created as follows:

```

proc access dbms=Datcom ad=mylib.custs;
  create vlib.cusphon.view;

```

```

select customer telephone name;
list view;
run;

```

---

### **VLIB.CUSTADD View Descriptor**

The VLIB.CUSTADD view descriptor was created as follows:

```

proc access dbms=Datacom ad=mylib.custs;
  create vlib.custadd.view;
  select state zipcode country name city;
  list view;
run;

```

---

### **VLIB.DCMEMPS View Descriptor**

The VLIB.DCMEMPS view descriptor was created as follows:

```

proc access dbms=Datacom ad=mylib.employee;
  create vlib.dcmemps.view;
  select empid birthdate lastname firstname
         middlename;
  list view;
run;

```

---

### **VLIB.EMPINFO View Descriptor**

The VLIB.EMPINFO view descriptor was created as follows:

```

proc access dbms=Datacom ad=mylib.employee;
  create vlib.empinfo.view;
  select empid dept lastname;
  list view;
run;

```

---

### **VLIB.EMPS View Descriptor**

The VLIB.EMPS view descriptor was created as follows:

```

proc access dbms=Datacom ad=mylib.employee;
  create vlib.emps.view;
  select empid jobcode birthdate lastname;
  subset where jobcode = 602;
  subset sort lastname;
  list view;
run;

```

---

### **VLIB.FORINV View Descriptor**

The VLIB.FORINV view descriptor was created as follows:

```

proc access dbms=Datacom ad=mylib.invoice;
  create vlib.forinv.view;

```

```
select all;
subset where country != 'USA';
list view;
run
```

---

## **VLIB.INV View Descriptor**

The VLIB.INV view descriptor was created as follows:

```
proc access dbms=Datacom ad=mylib.invoice;
create vlib.inv.view;
select invoicenum amtbilled country billedby paidon;
subset sort billedby;
list view;
run;
```

---

## **VLIB.USACUST View Descriptor**

The VLIB.USACUST view descriptor was created as follows:

```
proc access dbms=Datacom ad=mylib.custs;
create vlib.usacust.view;
select customer state zipcode name
       firstorderdate;
list view;
extend view;

subset where customer eq 1#;
subset sort firstorderdate;
list view;
list all;
run;
```

---

## **VLIB.USAINV View Descriptor**

The VLIB.USAINV view descriptor was created as follows:

```
proc access dbms=Datacom ad=mylib.invoice;
create vlib.usainv.view;
select all;
subset where country = 'USA';
list view;
run;
```

---

## VLIB.USAORDR View Descriptor

The VLIB.USAORDR view descriptor was created as follows:

```
proc access dbms=Datacom ad=mylib.orders;
  create vlib.usaordr.view;
  select ordernum stocknum length fabriccharges shipto;
  subset sort shipto;
  subset where shipto = 1#;
  list view;
run;
```

---

## SAS Data Files Used for CA-Datacom/DB Examples

---

### MYDATA.OUTOFSTK Data File

The SAS data file MYDATA.OUTOFSTK is used in Chapter 4, “Using CA-Datacom/DB Data in SAS Programs,” on page 23. It was created with the following SAS statements:

```
libname mydata 'your-SAS-library';
data mydata.outofstk;
  input fibernam $8. fibernum;
  datalines;
olefin  3478
gold    8934
dacron  4789
;
```

The following PRINT procedure produces the report shown in the output that follows.

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

#### Output A3.5 SAS Data File MYDATA.OUTOFSTK

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

---

## MYDATA.SASEMPS Data File

The SAS data file MYDATA.SASEMPS is used in Chapter 5, “Browsing and Updating CA-Datcom/DB Data,” on page 43. It was created with the following SAS statements:

```
libname mydata 'your-SAS-library';
data mydata.sasemps;
  input empid birthdat date7. lastname $18. firstnam $15.
        middlena $15.;
  datalines;
245962 30AUG64 BEDORTHA           KATHY           MARTHA
765432 01MAR59 POWELL             FRANK           X.
219223 13JUN47 HANSINGER         BENJAMIN        HAROLD
326745 21FEB52 RAWN             BEATRICE        MAY
;
```

The following PRINT procedure produces the report shown in the output that follows.

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

### Output A3.6 SAS Data File 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

---

## LIB6.BIRTHDAY Data File

The SAS data file LIB6.BIRTHDAY is used in Chapter 4, “Using CA-Datcom/DB Data in SAS Programs,” on page 23. It was created with the following SAS statements:

```
libname lib6 'your-SAS-library';
data lib6.birthday;
  input empid birthdat date7. 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
;
```

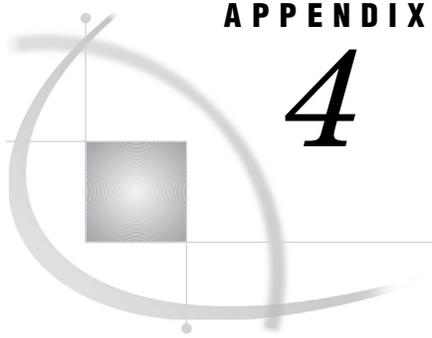
The following PRINT procedure produces the report shown in the following output.

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

**Output A3.7** SAS Data File LIB6.BIRTHDAY

LIB6.BIRTHDAY Data File			
OBS	EMPID	BIRTHDAT	LASTNAME
1	129540	31JUL60	CHOLAI
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





## Recommended Reading

---

*Recommended Reading* 153

---

### 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 z/OS Environment*
- SAS/CONNECT User's Guide*
- SAS/GRAPH Software: Reference, Volumes 1 and 2*
- SAS/STAT User's Guide, Volumes 1, 2, and 3*

For a complete list of SAS publications, go to [support.sas.com/bookstore](http://support.sas.com/bookstore). If you have questions about which titles you need, please contact a SAS Publishing Sales Representative at:

SAS Publishing Sales  
SAS Campus Drive  
Cary, NC 27513  
Telephone: 1-800-727-3228  
Fax: 1-919-531-9439  
E-mail: [sasbook@sas.com](mailto:sasbook@sas.com)  
Web address: [support.sas.com/bookstore](http://support.sas.com/bookstore)

Customers outside the United States and Canada, please contact your local SAS office for assistance.



# Glossary

---

**access descriptor**

a SAS/ACCESS file that describes data that is managed by a data management system. After creating an access descriptor, you can use it as the basis for creating one or more view descriptors. See also SAS data view, view descriptor.

**area**

an entity-type that consists of a collection of information in the CA-DATADITIONARY 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 personal computer 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. Depending on how the file is accessed, you can view SAS data either one observation (row) at a time or as a group in a tabular format. You cannot update data that you are browsing.

**CA-DATADITIONARY database**

a collection of CA-DATACOM/DB tables that have been organized within certain CA-DATACOM/DB areas and files. The records in each table contain one or more FIELD entity-occurrences.

**column**

in relational databases, a vertical component of a table. Each column has a unique name, contains data of a specific type, and has certain attributes. A column is analogous to a variable in SAS terminology. In CA-DATACOM/DB, columns are created from fields.

**compound field**

a field that contains two or more simple fields or compound fields that are contiguous. A compound field can contain data of different types and lengths. Compound fields can also be repeated or can be contained by other fields. See also simple field.

**data management system**

an integrated software application that enables you to create and manipulate data in the form of databases.

**data type**

in a CA-DATACOM/DB database, a classification according to the representation of the values to be stored. The data type is an attribute of every field. It tells CA-DATACOM/DB how much physical storage to set aside for the field and the type of data the field will contain. CA-DATACOM/DB allows 16 different types of character and numeric data. The CA-DATACOM/DB data type is similar to the type attribute of SAS variables.

**data value**

in CA-DATACOM/DB, a character value or numeric value that is stored in a field.

**database**

an organized collection of related data. A database usually contains named files, named objects, or other named entities such as tables, views, and indexes. In CA-DATACOM/DB, a database is an entity-type that contains areas, files, records, and fields. Each DATABASE entity-occurrence has a name and attributes in the CA-DATADictionary database.

**database management system (DBMS)**

a software application that enables you to create and manipulate data that is stored in the form of databases. See also relational database management system.

**descriptor file**

a type of SAS/ACCESS file that is used to establish a connection between SAS and files that are created and maintained by other software applications. Descriptor files describe data to SAS. To create descriptor files, you use the ACCESS procedure. There are two types of descriptor files: access descriptors and view descriptors. See also access descriptor, view descriptor.

**editing data**

the process of viewing the contents of a file with the intent and the ability to change those contents. Depending on how the file is accessed, you can view the data either one observation at a time or in a tabular format.

**element**

a CA-DATACOM/DB unit of transfer between application programs and CA-DATACOM/DB. An element consists of one or more contiguous CA-DATACOM/DB fields.

**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 file format. There are several types of engines. See also interface library engine, library engine, native library engine, view engine.

**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 file format. There are several types of engines. See also interface view engine.

**entity-occurrence**

an instance of a particular entity-type. For example, defining a database involves storing information about the database in a DATABASE occurrence. Each database is listed by its unique name as an occurrence of the DATABASE entity.

**entity-type**

any of the following data structures in which CA-DATACOM/DB data can be stored: databases, areas, fields, records, and elements.

**field**

the vertical component of a CA-DATACOM/DB table, which other software vendors refer to as a column or as a variable. There are four types of fields: key field, simple field, compound field, and repeating field. Each field has a name as well as specific attributes such as data type and length.

**file**

In CA-DATACOM/DB, each database contains one or more FILE entity- occurrences that comprise specific records, fields, and elements. Each FILE entity-occurrence requires a unique name and specific attributes in the CA-DATADITIONARY database.

**format, column**

an instruction that SAS uses to display or write each value of a variable. Some formats are supplied by SAS software. You can create your own formats by using the FORMAT procedure in Base SAS software.

**index**

in other software vendors' databases, a named object that directs the DBMS to the storage location of a particular data value for a particular column. Some DBMSs have additional specifications. These indexes are also used to optimize the processing of WHERE clauses and joins. Depending on the SAS interface to a database product and how selection criteria are specified, SAS might be able to use the indexes of the DBMS to speed data retrieval. In CA-DATCOM/DB, an index contains an entry for each key value in each record in a database. If a field is not indexed, the values are not indexed. However, the fields and key values can still be searched sequentially.

**informat, column**

an instruction that SAS uses to read raw data values to create variable values. Some informats are supplied by SAS software. Other informats can be written by the user with the FORMAT procedure in Base SAS software.

**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 have been formatted by another vendor's software and which presents the data to SAS in the form of a SAS data set. Interface view engines are transparent to users and are not specified in LIBNAME statements. See also engine.

**key**

in CA-DATACOM/DB, a field that enables you to quickly select and sequence data records. A key can be any combination of simple and compound fields and can be up to 180 characters long. The fields in the key do not have to be contiguous. See also Native Key, Master Key.

**Master Key**

a field that enables you to prevent values in a key field from being duplicated and to prevent values in that key from being changed. Each record must have one Master Key. The Master Key can also be the Native Key.

**member**

a SAS file in a SAS library.

**member name**

a name that is assigned to a SAS file in a SAS library. See also member type.

**member type**

a SAS name that identifies the type of information that is stored in a SAS file. Member types include ACCESS, AUDIT, DMBD, DATA, CATALOG, FDB, INDEX, ITEMSTOR, MDDB, PROGRAM, UTILITY, and VIEW.

**missing value**

in a CA-DATACOM/DB database, missing values are always represented by a blank space.

**Multi-User environment**

a CA-DATACOM/DB execution environment in which several users access a database at the same time, with queries and updates being handled simultaneously by a single copy of the software. See also single-user environment.

**Native Key**

a field that determines the order of the records in a CA-DATACOM/DB table. Each table must have one Native Key. The Native Key can also be the Master Key.

**record**

the horizontal component of a CA-DATACOM/DB table. A record is a set of fields that are treated as a unit. Records within a table are ordered by the Native Key. A record is analogous to a SAS row.

**relational database management system**

a database management system that organizes and accesses data according to relationships between data items. The main characteristic of a relational database management system is the two-dimensional table. Examples of relational database management systems are DB2, Oracle, Sybase, and Microsoft SQL Server.

**repeating field**

a simple field or compound field that can occur more than once. Repeating fields can be nested within other repeating fields.

**row**

a collection of data values that are associated with a single entity such as a customer or a state. Each row contains one data value for each variable.

**SAS data file**

a type of SAS data set that contains data values as well as descriptor information that is associated with the data. The descriptor information includes information such as the data types and lengths of the variables, as well as the name of the engine that was used to create the data. See also SAS data set, SAS data view.

**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**

a type of SAS data set that retrieves data values from other files. A SAS data view contains only descriptor information such as the data types and lengths of the variables (columns), plus other information that is required for retrieving data values from other SAS data sets or from files that are stored in other software vendors' file formats. SAS data views can be created by the ACCESS and SQL procedures, as well as by the SAS DATA step.

**simple field**

a single CA-DATACOM/DB field. See also compound field.

**single-user environment**

a CA-DATACOM/DB execution environment in which each user has a copy of CA-DATACOM/DB and has exclusive use of any database that he or she is currently accessing. See also multi-user environment.

**table**

In CA-DATACOM/DB, the combination of one FILE entity-occurrence and one RECORD entity-occurrence describes a table.

**URT (User Requirements Table)**

See User Requirements Table (URT).

**user ID**

a string of characters that a user must specify correctly in order to gain access to the CA-DATADictionary database.

**User Requirements Table (URT)**

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

**variable**

In the ACCESS procedure, variables are created from the database product's columns or fields.

**view**

a generic term (used by many software vendors) for a definition of a virtual data set (or table). The definition is named and stored for later use. A view contains no data; it merely describes or defines data that is stored elsewhere. See also SAS data view.

**view descriptor**

The subset consists of selected records in a CA-DATACOM/DB table.

**windowing procedure**

a SAS procedure that you can use by entering information in one or more windows or dialog boxes. For example, the FSVIEW procedure is a windowing procedure. Some procedures, such as ACCESS and DBLOAD, can be used either as windowing procedures or in batch mode.



# Index

---

## A

- ACCDESC= option
  - ACCESS procedure (CA-Datcom/DB) 61, 62
- access by row number 111
- access descriptors 4, 5, 15
  - assigning passwords 63
  - CA-DATADictionary changes and 109
  - creating 16, 65, 69
  - creating view descriptors from 65, 67
  - passwords 63
  - resetting column defaults 83
  - updating 20, 65, 87
- ACCESS procedure, CA-Datcom/DB 61
  - calls to interface view engine 104
  - creating descriptor files 16
  - data conversions 97
  - efficient view descriptors 96
  - extracting data 21
  - invoking 65
  - passwords for descriptors 63
  - SORT clause in view descriptors 95
  - syntax 61
  - terminating 82
  - WHERE clause in view descriptors 89
- APPEND procedure 53
- appending data 53
- ASSIGN statement
  - ACCESS procedure (CA-Datcom/DB) 67
- ASSIGN\_statement 67
- asterisk (\*)
  - in view WHERE clauses 91

## B

- browsing data 43
  - FSBROWSE procedure 44
  - FSVIEW procedure 45
  - SELECT statement 50
  - SQL procedure 50
  - WHERE clause while browsing 47

## C

- CA-Datcom/DB 3, 7
- CA-Datcom/DB databases 9, 70
  - fast-loading process 120
  - status or version of 71

- CA-DATADictionary 7
  - passwords for 122
  - updates, and descriptor files 109
  - user ID 89, 123
  - validation against 126
- character data types 11
- CHART procedure 26
- charting data 26
- columns 23, 24
  - changing format of 73
  - changing informat of 74
  - dropping 72
  - listing 72, 76
  - naming conventions 67
  - renaming 82
  - resetting to defaults 83
  - selecting for view descriptors 84
- combining data
  - See* selecting and combining data
- COMMIT statement
  - row processing and 118
- compound fields 10
- concurrent tasking 112
- CONTENT statement
  - ACCESS procedure (CA-Datcom/DB) 68
- CONTENT\_statement 68
- CREATE statement
  - ACCESS procedure (CA-Datcom/DB) 69
- CREATE\_statement 69

## D

- data conversions 97
- data fields
  - listing 77
- data files
  - example data 149
  - updating 38
- data set options
  - CA-Datcom/DB 117
- data sets
  - passwords 63
- data types 10
- data validation 126
- data views 8
- database administration 103
  - access by row number 111
  - CA-Datcom/DB interface 104
  - debugging 113

- direct addressing 111
  - error messages 113
  - interface view engine 104
  - locks 111
  - multi-tasking 112
  - password encryption 112
  - performance 112
  - recovery processing 108
  - retrieval processing 105
  - security 110
  - spool files 111
  - system options for 113
  - update processing 107
  - user requirements table 110
  - DATABASE entity-type 9
  - DATABASE statement
    - ACCESS procedure (CA-Datcom/DB) 70
  - databases (CA-Datcom/DB)
    - See CA-Datcom/DB databases
  - DATABASE\_statement 70
  - DATASETS procedure
    - assigning passwords 64
  - date format 68
  - date length 68
  - date types 11
  - DBMS= option
    - ACCESS procedure (CA-Datcom/DB) 62
  - DBSTAT statement
    - ACCESS procedure (CA-Datcom/DB) 71
  - DBSTAT\_statement 71
  - DDBCOMIT= data set option 118
  - DDBCOMIT\_data\_set\_option 118
  - DDBDBN system option 114
  - DDBDELIM system option 114
  - DDBERLMT= data set option 119
  - DDBERLMT\_data\_set\_option 119
  - DDBKEY= data set option 120
  - DDBKEY\_data\_set\_option 120
  - DDBLOAD= data set option 120
  - DDBLOAD system option 114
  - DDBLOAD\_data\_set\_option 120
  - DDBLOCK= data set option 121
  - DDBLOCK system option 114
  - DDBLOCK\_data\_set\_option 121
  - DDBMASK system option 114
  - DDBMISS system option 114
  - DDBPW= data set option 122
  - DDBPW system option 114
  - DDBPW\_data\_set\_option 122
  - DDBSPANS system option 114
  - DDBSV= data set option 122
  - DDBSV system option 114
  - DDBSV\_data\_set\_option 122
  - DDBTASK system option 114
  - DDBTRACE= data set option 113, 123
  - DDBTRACE system option 113, 114
  - DDBTRACE\_data\_set\_option 122
  - DDBUPD system option 114
  - DDBURT= data set option 123
  - DDBURT system option 114
  - DDBURT\_data\_set\_option 123
  - DDBUSER= data set option 123
  - DDBUSER system option 114
  - DDBUSER\_data\_set\_option 123
  - debugging 113, 123
  - Default Keys 120
  - DELETE statement
    - SQL procedure 53
  - deleting records
    - SAS/FSP procedures for 48
  - descriptor files 4, 15
    - CA-DATADITIONARY changes and 109
    - creating 16, 69
    - creating, in one PROC step 16
    - defining 15
    - updating 20, 87
  - direct addressing 111
  - DROP statement
    - ACCESS procedure (CA-Datcom/DB) 72
  - DROP\_statement 71
- ## E
- elements 10
  - encryption 112
  - entity-occurrence 8
  - entity-types 8
    - DATABASE 9
    - FIELD 10
    - RECORD 9
  - error messages 113
  - example data 6, 129
    - SAS data files 149
    - tables 130
  - execution environments 13
  - EXTEND statement
    - ACCESS procedure (CA-Datcom/DB) 72
  - EXTEND\_statement 72
  - extracting data 21, 41
- ## F
- fast-loading process 120
  - FIELD entity-type 10
  - fields
    - creating 37
    - data types in 10
  - FORMAT statement
    - ACCESS procedure (CA-Datcom/DB) 73
  - formats
    - changing, for columns 73
  - FORMAT\_statement 73
  - FREQ procedure 28
  - FSBROWSE procedure 44
  - FSEDIT procedure 45
  - FSVIEW procedure
    - browsing data 45
    - internal record ID and 107
    - updating data 46
- ## G
- GETIT command 106
  - GETPS command 106
  - GROUP BY clause
    - creating fields 37
  - GSETL command 106
  - GSETP command 106
- ## H
- \$HEX. format 92

**I**

index reads 42  
indexing 12  
INFORMAT statement  
    ACCESS procedure (CA-Datcom/DB) 74  
informats  
    changing, for columns 74  
INFORMAT\_statement 74  
INSERT statement  
    SQL procedure 53  
inserting records 124  
INSERT statement (SQL) 53  
SAS/FSP procedures for 48  
interface to CA-Datcom/DB 3, 7  
interface view engine 3  
    calls to 104  
    database administration 104  
    URT specification for 123  
internal record ID (RID) 106

**K**

KEEP= data set option 31  
key fields 10, 75  
KEY statement  
    ACCESS procedure (CA-Datcom/DB) 75  
keys  
    Default Keys 120  
KEY\_statement 75

**L**

LIST statement  
    ACCESS procedure (CA-Datcom/DB) 76  
LISTINFO statement  
    ACCESS procedure (CA-Datcom/DB) 77  
LISTINFO\_statement 77  
LISTOCC statement  
    ACCESS procedure (CA-Datcom/DB) 78  
LISTOCC\_statement 77  
LIST\_statement 76  
LOCKG command 106  
locking records 111  
LOCKL command 106  
LOCNX command 106

**M**

masking values 93  
Master Key 10  
MEANS procedure 28  
member-level locking 111  
missing values 11  
    in tables 125  
multi-field keys  
    in view WHERE clauses 94  
multi-tasking 112  
multi-user environment 13

**N**

Native Key 10  
nils 11  
    in tables 125  
numeric data types 10

**O**

OBS= option  
    PRINT procedure 25  
OCCURS 78  
OCCURS statement  
    ACCESS procedure (CA-Datcom/DB) 79  
OUT= option  
    ACCESS procedure (CA-Datcom/DB) 62

**P**

PASSWORD statement  
    ACCESS procedure (CA-Datcom/DB) 81  
passwords  
    access descriptors 63  
    CA-DATADictionary 122  
    data sets 63  
    encryption 112  
    specifying 81  
PASSWORD\_statement 81  
percentages 28  
performance 41, 112  
PRINT procedure 25  
    OBS= option 25  
printing data 25  
PROC ACCESS statement  
    CA-Datcom/DB 61, 62

**Q**

QUIT statement  
    ACCESS procedure (CA-Datcom/DB) 82  
QUIT\_statement 82

**R**

RANK procedure 30  
RDUBR command 106  
RDUKG command 106  
RDUKL command 106  
RDULE command 106  
RDUNX command 106  
RECORD entity-type 9  
record-level locking 111  
recovery processing 108  
REDLE command 106  
RENAME statement  
    ACCESS procedure (CA-Datcom/DB) 82  
RENAME\_statement 82  
repeating fields 10  
    DB content attributes 79  
    dropping occurrences 79  
    format attributes 80  
    informat attributes 80  
    listing 78  
    modifying occurrences of 79  
    renaming columns 80  
    resetting occurrences 80  
    selecting occurrences 81  
RESET statement  
    ACCESS procedure (CA-Datcom/DB) 83  
RESET\_statement 83  
retrieval processing 105  
    internal record ID 106  
    no WHERE clause 105

- SORT clause with 105
- WHERE clause with 105
- RID (internal record ID) 106
- rollback 119
- row number access 111
- row processing
  - before COMMIT statement 118
- rows 23

## S

- SAS/ACCESS interface to CA-Datcom/DB 3, 7
- SAS data files
  - example data 149
  - updating 38
- SAS/FSP procedures
  - browsing and updating data 44
  - inserting and deleting records 48
- security 13, 110
- SELECT statement
  - ACCESS procedure (CA-Datcom/DB) 84
  - SQL procedure 50
- selecting and combining data 30
  - SQL procedure 32
  - WHERE statement 31
- selection criteria 126
  - SAS WHERE clause 126
  - view descriptors 85
  - WHERE clause in view descriptor 126
- SELECT\_statement 84
- simple fields 10
- single user environment 13
- SORT clauses 12
  - retrieval processing with 105
  - view SORT clauses 95
- sorting data 12, 42
- spool files 111
  - suppressing 121
- SQL procedure
  - browsing and updating data 50
  - combining data 32
  - creating fields 37
  - DELETE statement 53
  - GROUP BY clause 37
  - INSERT statement 53
  - SELECT statement 50
  - UNDO\_POLICY option 50
  - UPDATE statement 52
- statistics calculation 28
- SUBSET statement
  - ACCESS procedure (CA-Datcom/DB) 85
- SUBSET\_statement 85
- subsetting data 12
- system options
  - CA-Datcom/DB 113

## T

- table locks 111
- TABLE statement
  - ACCESS procedure (CA-Datcom/DB) 86
- tables 9
  - altering, and descriptor files 66
  - deleting records 124
  - example data 130
  - inserting records 124

- missing values in 125
- specifying 86
- status of 86, 122
- version of 86, 122
- TABLE\_statement 86
- TBLSTAT statement
  - ACCESS procedure (CA-Datcom/DB) 86
- TBLSTAT\_statement 86
- transaction data 38

## U

- UNDO\_POLICY option
  - SQL procedure 50
- update processing 107
  - repositioning to inserted record 107
- UPDATE statement
  - ACCESS procedure (CA-Datcom/DB) 87
  - SQL procedure 52
- UPDATE\_statement 87
- updating access descriptors 20, 65, 87
- updating data 43
  - FSEDIT procedure 45
  - FSVIEW procedure 46
  - SQL procedure 50
  - UNDO\_POLICY option (SQL) 50
  - UPDATE statement (SQL) 52
  - WHERE clause while updating 47
- updating descriptor files 20, 87
- updating SAS data files 38
- updating view descriptors 20, 66, 87
- URT statement
  - ACCESS procedure (CA-Datcom/DB) 89
- URT (user requirements table) 89, 110, 123
- URT\_statement 88
- user exits
  - CA-Datcom/DB 124
- user IDs 89
- user requirements table (URT) 89, 110, 123
- USER statement
  - ACCESS procedure (CA-Datcom/DB) 89
- USER\_statement 89

## V

- validating data values 126
- validation against CA-DATADICIONARY 126
- Version 6 files 38
- Version 8 (and later) files 40
- view descriptors 4, 5, 15
  - CA-DATADICIONARY changes and 109, 110
  - creating 16, 65, 69, 70
  - creating from access descriptors 65, 67
  - dropping columns 72
  - efficient use of 96
  - multiple 124
  - referencing 42
  - resetting column defaults 83
  - selecting columns for 84
  - selection criteria for 85
  - SORT clause in 95
  - sorting data in 12
  - updating 20, 66, 87
  - WHERE clauses in 89, 126
- view SORT clauses 95

- view WHERE clauses 89
  - asterisk (\*) in 91
  - character fields 92
  - date values 92
  - expressions 91
  - guidelines for 94
  - \$HEX, format fields 92
  - masking values in 93
  - multi-field keys in 94
  - syntax 89
  - values mismatched to fields 93
- VIEWDESC= option
  - ACCESS procedure (CA-Datcom/DB) 62

## W

- WHERE clauses
  - in view descriptors 126
  - retrieval processing with 105
  - selection criteria 126
  - unacceptable conditions 125
  - view WHERE clauses 89
  - while browsing or updating data 47
- WHERE statement
  - selecting data 31



# Your Turn

---

We want your feedback.

- If you have comments about this book, please send them to **[yourturn@sas.com](mailto:yourturn@sas.com)**. Include the full title and page numbers (if applicable).
- If you have comments about the software, please send them to **[suggest@sas.com](mailto:suggest@sas.com)**.



# SAS® Publishing delivers!

Whether you are new to the workforce or an experienced professional, you need to distinguish yourself in this rapidly changing and competitive job market. SAS® Publishing provides you with a wide range of resources to help you set yourself apart.

## **SAS® Press Series**

Need to learn the basics? Struggling with a programming problem? You'll find the expert answers that you need in example-rich books from the SAS Press Series. Written by experienced SAS professionals from around the world, these books deliver real-world insights on a broad range of topics for all skill levels.

**[support.sas.com/saspress](http://support.sas.com/saspress)**

## **SAS® Documentation**

To successfully implement applications using SAS software, companies in every industry and on every continent all turn to the one source for accurate, timely, and reliable information—SAS documentation. We currently produce the following types of reference documentation: online help that is built into the software, tutorials that are integrated into the product, reference documentation delivered in HTML and PDF—free on the Web, and hard-copy books.

**[support.sas.com/publishing](http://support.sas.com/publishing)**

## **SAS® Learning Edition 4.1**

Get a workplace advantage, perform analytics in less time, and prepare for the SAS Base Programming exam and SAS Advanced Programming exam with SAS® Learning Edition 4.1. This inexpensive, intuitive personal learning version of SAS includes Base SAS® 9.1.3, SAS/STAT®, SAS/GRAPH®, SAS/QC®, SAS/ETS®, and SAS® Enterprise Guide® 4.1. Whether you are a professor, student, or business professional, this is a great way to learn SAS.

**[support.sas.com/LE](http://support.sas.com/LE)**



**THE  
POWER  
TO KNOW®**

