



THE
POWER
TO KNOW.

SAS/ACCESS[®] 9.3 Interface to SYSTEM 2000[®] Reference

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2011. *SAS/ACCESS® 9.3 Interface to SYSTEM 2000®: Reference*. Cary, NC: SAS Institute Inc.

SAS/ACCESS® 9.3 Interface to SYSTEM 2000®: Reference

Copyright © 2011, SAS Institute Inc., Cary, NC, USA

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.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a) and DFAR 227.7202-4 and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513-2414.

July 2015

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 trademarks of their respective companies.

Contents

Chapter 1 • The SAS/ACCESS Interface to SYSTEM 2000	1
Overview of the SAS/ACCESS Interface to SYSTEM 2000	1
Example Data in This Document	2
Chapter 2 • SYSTEM 2000 Software	5
Overview of SYSTEM 2000	5
SYSTEM 2000 Databases	6
SYSTEM 2000 Item Types	12
SYSTEM 2000 Indexing	14
Selecting a Subset of Data	14
Sorting Output	15
SYSTEM 2000 Passwords	15
SYSTEM 2000 Execution Environments	16
SYSTEM 2000 Database Files	16
Chapter 3 • SAS/ACCESS Descriptor Files	19
Overview of SAS/ACCESS Descriptor Files	19
Defining SAS/ACCESS Descriptor Files	19
Creating Descriptor Files	20
Chapter 4 • SYSTEM 2000 Data in SAS Programs	23
Using SYSTEM 2000 Data in SAS	23
Reviewing Variables	24
Printing Data	25
Charting Data	26
Calculating Statistics	27
Selecting and Combining Data with the SQL Procedure	30
Updating SAS Data Files with SYSTEM 2000 Data	35
Performance Considerations	38
Chapter 5 • Browsing and Updating SYSTEM 2000 Data	41
Browsing and Updating SYSTEM 2000 Data Directly from SAS	41
Browsing and Updating with SAS/FSP	42
Browsing and Updating with the SQL Procedure	49
Using the APPEND Procedure	52
Browsing and Updating with the QUEST Procedure	56
Chapter 6 • Creating and Loading SYSTEM 2000 Databases	59
DBLOAD Procedure in SAS and SYSTEM 2000	59
Selecting a Processing Mode for Loading Data	66
Chapter 7 • SAS/ACCESS 9 for SYSTEM 2000: Reference	67
ACCESS Procedure in SAS and SYSTEM 2000	68
ACCESS Procedure Statements	70
ACCESS Procedure Syntax	70
The PROC ACCESS Statement	71
where-clause in SYSTEM 2000	72
ordering-clause in SYSTEM 2000	77
Creating and Using View Descriptors Efficiently	78
PROC ACCESS Data Conversions	78
Dictionary	79

Chapter 8 • DBLOAD Procedure Reference	91
DBLOAD Procedure and SYSTEM 2000	91
Creating Customized View Descriptors	92
Default SYSTEM 2000 Item Types	92
Allocating the Database Files	93
Adding Disjoint Schema Records	94
Loading One SYSTEM 2000 Database from Another	94
DBLOAD Procedure Options Syntax	94
Dictionary	95
Chapter 9 • QUEST Procedure Reference	105
QUEST Procedure in SAS with SYSTEM 2000	105
Statements in PROC QUEST	106
SYSTEM 2000 Statements and the QUEST Procedure	106
Single-User Mode	108
Multi-User Mode	109
Dictionary	110
Appendix 1 • Topics for Database Administrators	113
SYSTEM 2000 and the SAS/ACCESS Interface	113
Changing a SYSTEM 2000 Database Password	116
Changing a Database Definition	116
Data Security	117
Enabling the Rollback Log	118
Locking Record Levels	119
Maximizing SYSTEM 2000 Performance	120
Appendix 2 • Advanced Topics for Users	121
Overriding Options	121
Using Multiple View Descriptors	122
Deleting Data Records	122
Inserting Data Records	123
BY Key	123
Missing Values (Nulls)	126
WHERE Clauses in SAS and where-clauses in SYSTEM 2000	127
Specifying Selection Criteria	133
Connecting Strings to Order Conditions	133
Stored Strings in SYSTEM 2000	136
Appendix 3 • Example Programs	137
Using the Example Programs	137
SYSTEM 2000 Database Definition for Database EMPLOYEE	137
Access Descriptors	139
View Descriptors	141
SAS Data Files	152
Recommended Reading	157
Glossary	159
Index	181

Chapter 1

The SAS/ACCESS Interface to SYSTEM 2000

Overview of the SAS/ACCESS Interface to SYSTEM 2000	1
Example Data in This Document	2

Overview of the SAS/ACCESS Interface to SYSTEM 2000

SAS/ACCESS software provides an interface between SAS and SYSTEM 2000 database management software. With the SAS/ACCESS interface, you can do the following:

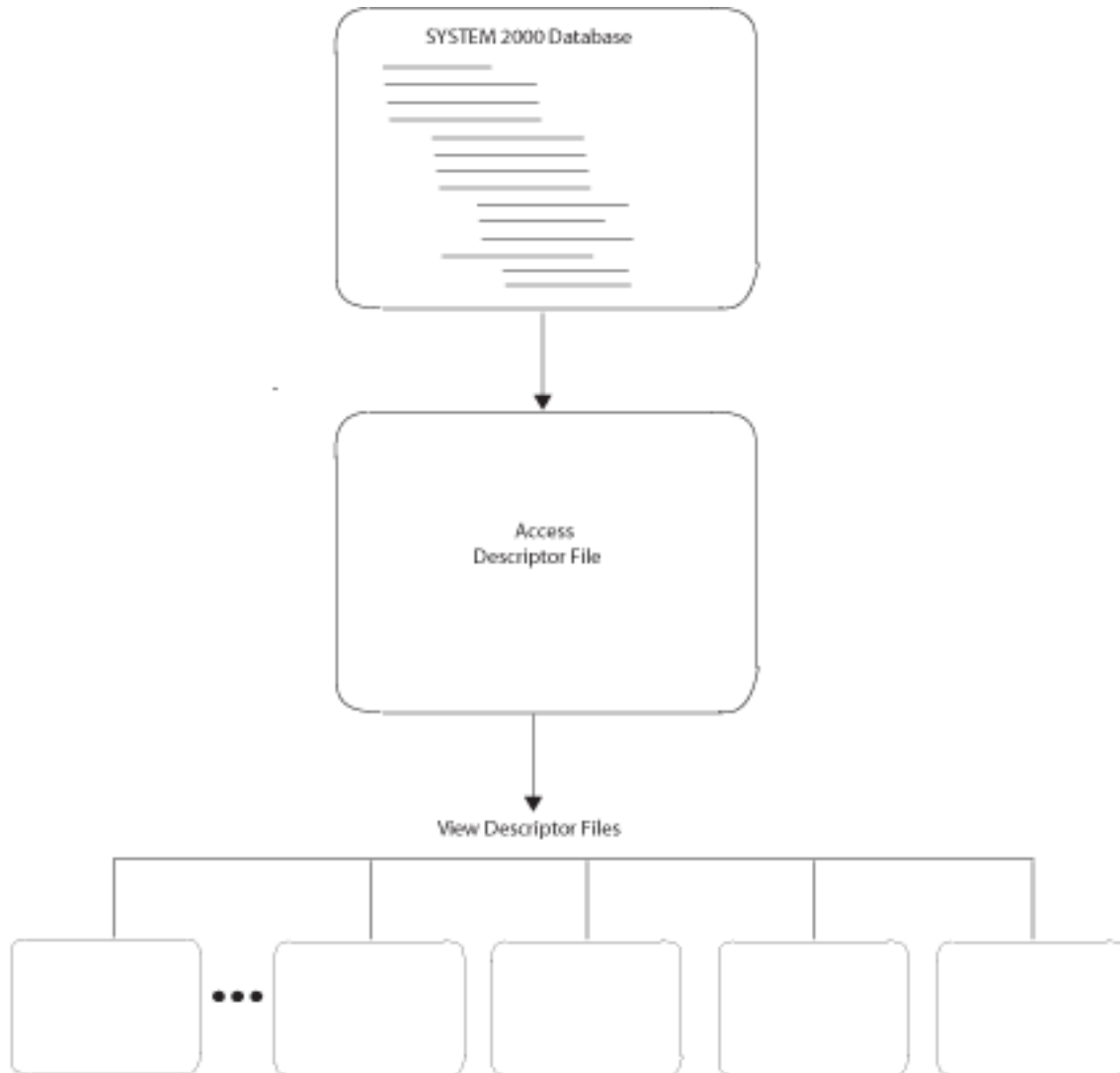
- create SAS/ACCESS descriptor files by using the ACCESS procedure.
- directly access data in SYSTEM 2000 databases from within a SAS program by using the descriptor files created with the ACCESS procedure.
- extract SYSTEM 2000 data and place it in a SAS data file by using the ACCESS procedure or the DATA step.
- load data into a SYSTEM 2000 database by using the DBLOAD procedure.
- update data in SYSTEM 2000 databases by using the SQL procedure, SAS/FSP software, or the APPEND procedure. The SQL procedure can also be used to join SAS data files, PROC SQL views, and SAS/ACCESS view descriptors.
- directly access data in SYSTEM 2000 databases by using the QUEST procedure with SYSTEM 2000 statements.

The SAS/ACCESS interface consists of the following:

- the interface view engine, which enables you to use SYSTEM 2000 data in SAS programs in the same way that you use SAS data. You can print, plot, and chart the data described by the descriptor files, use the descriptor files to create other SAS data files, and so on.
- the ACCESS procedure, which enables you to describe SYSTEM 2000 data to SAS and store the description in SAS/ACCESS descriptor files.
- the DBLOAD procedure, which enables you to create and load SYSTEM 2000 databases using data from SAS data sets.
- the QUEST procedure, which enables you to access SYSTEM 2000 and issue SYSTEM 2000 statements from SAS.

You might need to combine data from several SYSTEM 2000 databases or from external databases, such as DB2 or SAS 6 and later SAS data sets. Such combinations are not only possible but easy to do. SAS can distinguish between SAS data files, SAS/ACCESS descriptor files, and other types of SAS files, and the software uses the appropriate access method.

The following figure illustrates the relationships of a SYSTEM 2000 database, access descriptors, and view descriptors.



Example Data in This Document

This document uses the SYSTEM 2000 database Employee and several SAS data files to show you how to use the SAS/ACCESS interface to SYSTEM 2000. This database and the SAS data files MyData.Classes, V6.Birthdy, and MyData.CorPhon were created for a company's employee information. The data file Trans.Banking (used in the example for the DBLOAD procedure) was created for banking transactions. All the data is fictitious. The database is used to show how the interface treats SYSTEM 2000 data. It should not be used as an example for you to follow in designing databases for any purpose.

See [Appendix 3, “Example Programs,” on page 137](#) for more information about the example data used in this documentation.

The SAS jobs to create the SAS data files are on your installation media; see your on-site SAS support personnel to create the SAS data files. The database Employee is on the SYSTEM 2000 installation media; see your Database Administrator to ensure that the data is available and in its original state.

You create the access descriptor MyLib.Employe and the view descriptors Vlib.EmpPos and Vlib.EmpSkil. See [SAS/ACCESS Descriptor Files on page 19](#). You need to create the other view descriptors in this document on your own, using the definitions shown in [Appendix 3, “Example Programs,” on page 137](#).

Chapter 2

SYSTEM 2000 Software

Overview of SYSTEM 2000	5
SYSTEM 2000 Databases	6
Overview of Database Definition	6
Database Names	7
Labeling Data	7
Grouping Schema Items	8
Grouping Schema Records	8
Logical Entries	10
Mapping Data between SAS and SYSTEM 2000	11
Null Data (Missing Values)	12
SYSTEM 2000 Item Types	12
Overview of Item Types	12
Numeric Item Types	13
Date Item Types	13
Character Item Types	13
SYSTEM 2000 Indexing	14
Selecting a Subset of Data	14
Sorting Output	15
SYSTEM 2000 Passwords	15
SYSTEM 2000 Execution Environments	16
SYSTEM 2000 Database Files	16

Overview of SYSTEM 2000

Note: This section focuses on terms and basic concepts that help you use the SAS/ACCESS interface to SYSTEM 2000. Experienced users of SYSTEM 2000 might want to proceed to [Chapter 3, “SAS/ACCESS Descriptor Files,”](#) on page 19.

SYSTEM 2000 is hierarchical database management software from SAS for mainframe computer systems that run under the z/OS operating environment. Using SYSTEM 2000, you can define the types of data to be stored in a database and the relationships in the data. You can also load the database and retrieve and update the data. The software's hierarchical database structure provides the following:

- efficient data storage by reducing the amount of redundant data

- indexing capabilities for fast and efficient retrievals
- data qualification and sorting capabilities
- complete data security through the use of passwords
- Multi-User and single-user execution environments
- a complete set of diagnostic messages
- optional transaction journaling
- rollback recovery from system crashes or other failures

SYSTEM 2000 Databases

Overview of Database Definition

A SYSTEM 2000 database is hierarchical because you can store and access data according to organized relationships in groups of associated data. When a SYSTEM 2000 database is created, a plan called a *database definition* is devised, in which the following happens:

- the database has an assigned name
- the data to be stored is labeled
- the data is arranged into groups
- relationships are established among the groups of data

Usually, a database is organized according to the types of data and the way you want to use the data. To create descriptor files for the SAS/ACCESS interface, you must understand and be familiar with the contents of the database and its organization in order to retrieve and update information accurately and efficiently. The output below shows an excerpt of the database definition for Employee, which is the output from a DESCRIBE statement in SYSTEM 2000. (For a complete listing of the database definition for Employee, see [Appendix 3, “Example Programs,”](#) on page 137.)

Output 2.1 Database Definition for the Database Employee

```

SYSTEM RELEASE NUMBER 12.1
DATA BASE NAME IS      EMPLOYEE
DEFINITION NUMBER      2
DATA BASE CYCLE NUMBER 25
  1* EMPLOYEE NUMBER (INTEGER NUMBER 9999)
  2* LAST NAME (CHAR X(10) WITH FEW FUTURE OCCURRENCES )
  3* FORENAME (NON-KEY CHAR X(20))
      .
      .
      .
16* ZIP CODE (CHAR X(5) WITH FEW FUTURE OCCURRENCES )
100* POSITION WITHIN COMPANY (RECORD)
101* POSITION TITLE (NON-KEY CHAR X(10) IN 100)
102* DEPARTMENT (CHAR X(14) IN 100 WITH SOME FUTURE OCCURRENCES )
103* MANAGER (CHAR XXX IN 100 WITH FEW FUTURE OCCURRENCES )
104* POSITION TYPE (CHAR X(12) IN 100 WITH SOME FUTURE OCCURRENCES )
105* START DATE (DATE IN 100)
106* END DATE (NON-KEY DATE IN 100)
110* SALARY WITHIN POSITION (RECORD IN 100)
  111* PAY RATE (MONEY $9999.99 IN 110)
  112* PAY SCHEDULE (CHAR X(7) IN 110)
  113* EFFECTIVE DATE (DATE IN 110)
  114* CURRENT DEDUCTION (NON-KEY MONEY $9999.99 IN 110)
      .
      .
      .
400* EDUCATIONAL BACKGROUND (RECORD)
410* EDUCATION (RECORD IN 400)
  411* SCHOOL (CHAR X(15) IN 410)
  412* DEGREE/CERTIFICATE (CHAR X(7) IN 410 WITH FEW FUTURE OCCUR
      RENCES )
  413* DATE COMPLETED (DATE IN 410)
  414* MAJOR FIELD (NON-KEY CHAR X(16) IN 410)
  415* MINOR FIELD (NON-KEY CHAR X(12) IN 410)

```

Database Names

The database name is a unique name, from 1 to 16 characters in length, that is assigned to a specific SYSTEM 2000 database definition. Each database also has one or more passwords associated with it.

To create descriptor files for the SAS/ACCESS interface, you must know the name and password for the SYSTEM 2000 database that you want to access.

Labeling Data

A database definition consists of schema items and schema records, which describe a blueprint for the type of data to be stored. For example, in the database definition for Employee, EMPLOYEE NUMBER is a schema item and POSITION WITHIN COMPANY is a schema record.

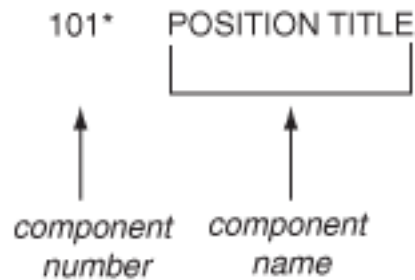
A *schema item* names and defines the characteristics of a group of values. A schema item has a name, a type, and a picture (length). Each value stored in a SYSTEM 2000 database corresponds to a schema item. For example, the following schema item describes the numbers used as employee identification numbers. The four 9s indicate that each employee number can contain a maximum of four digits.

```
1* EMPLOYEE NUMBER (INTEGER NUMBER 9999)
```

A *schema record* groups associated schema items. Schema records are explained in “Grouping Schema Items” on page 8.

Schema items and schema records are referred to as *schema components*, and each is identified by a component number and a component name, as shown in the following figure. (A component number can also be referred to as a C-number, such as C101.)

Figure 2.1 Schema Components



To access data stored in a SYSTEM 2000 database, you must specify either the component number or the component name. Both are unique in the database definition to avoid ambiguity. Each line in a database definition begins with the component number and the component name.

When you create descriptor files for SYSTEM 2000 databases, PROC ACCESS creates corresponding SAS variable names from the SYSTEM 2000 schema item names. You can then use the variable names in SAS procedures.

Grouping Schema Items

In a SYSTEM 2000 database, associated schema items are grouped by schema records. That is, different schema records store different groups of data, and a schema item belongs to only one schema record. Grouping associated schema items into schema records is similar to planning a form. A form is usually divided into sections with one section for each set of related data.

For example, look at the database definition for Employee in [Output 2.1 on page 7](#). Schema items C1 through C16 contain personal information about each employee. These items are grouped into one record, the ENTRY or C0 record. (The component number and name for the ENTRY record is not listed in a database definition unless it has been renamed.) The schema items C101 through C106, which contain information about an employee's position, are grouped in schema record C100, POSITION WITHIN COMPANY.

Grouping Schema Records

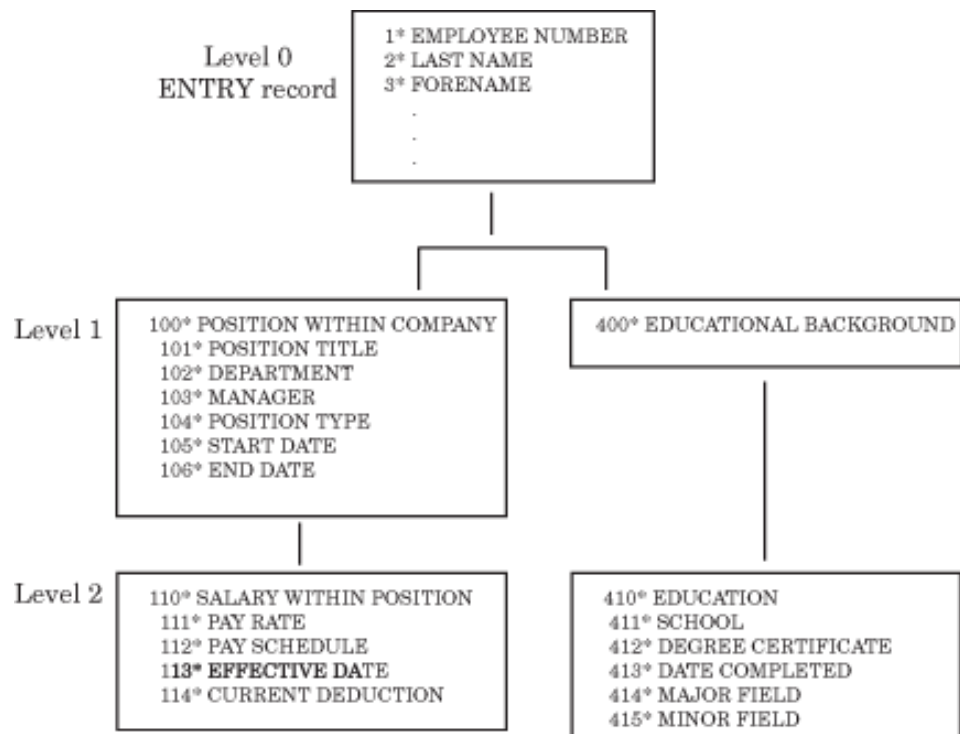
Schema record relationships are established by arranging the schema records into levels. Each schema record is placed at a specific level, which creates a hierarchical structure.

These levels are achieved by ranking schema items with values that occur only one time per employee (such as an employee's name and address) at a higher level than schema items with multiple values (such as an employee's job titles and salaries). That is, schema items having a one-to-many relationship with other schema items rank higher in the database hierarchy than the other schema items.

Look at the database definition for Employee shown in [Figure 2.2 on page 9](#). Notice that schema items are indented under their parent schema record, and schema records are indented farther to the right. This reflects the relationships among the records:

- Schema items C1 through C16 store values that occur only one time per employee and are grouped as the top level of the database in the ENTRY record or at *level 0*.
- The ENTRY record (C1 to C16) has a one-to-many relationship with the POSITION WITHIN COMPANY record (C100) because each employee can have more than one position during their employment, so position title, department, and so on, can have multiple values. Because positions are associated with specific employees, the POSITION WITHIN COMPANY record is related to the ENTRY record. POSITION WITHIN COMPANY is below level 0. It is at *level 1*.
- Positions have a one-to-many relationship with salary data because an employee can have more than one salary in a single position. Salary information is grouped in a record named SALARY WITHIN POSITION (C110), which is related to the POSITION WITHIN COMPANY record. SALARY WITHIN POSITION is below level 1. It is at *level 2*.

Figure 2.2 Levels in a Database Definition



The next set of terms refers to the relationships between the levels, which are like relationships in a family.

- A *parent* is the record immediately preceding a specified record. Each record can have only one parent, and no record is an orphan, except for the ENTRY record at level 0.
- An *ancestor* is a record that exists on the level that precedes a specified record in the same path. The ENTRY record is an ancestor of all other records in the database.
- A *descendant* is a record that exists at a lower level than a specified record in a family. C100 is a descendant of the ENTRY record; C110 is a descendant of the ENTRY record and C100.

- *Children* are the records immediately following a specified record. C100 is a child of the ENTRY record; C110 is a child of C100.
- A *family* consists of a record, all its ancestors, and all its descendants.
- The *path* of a record is a record and all its ancestors. C110, C100, and the ENTRY record make up a path; C410, C400, and the ENTRY record make up another path.

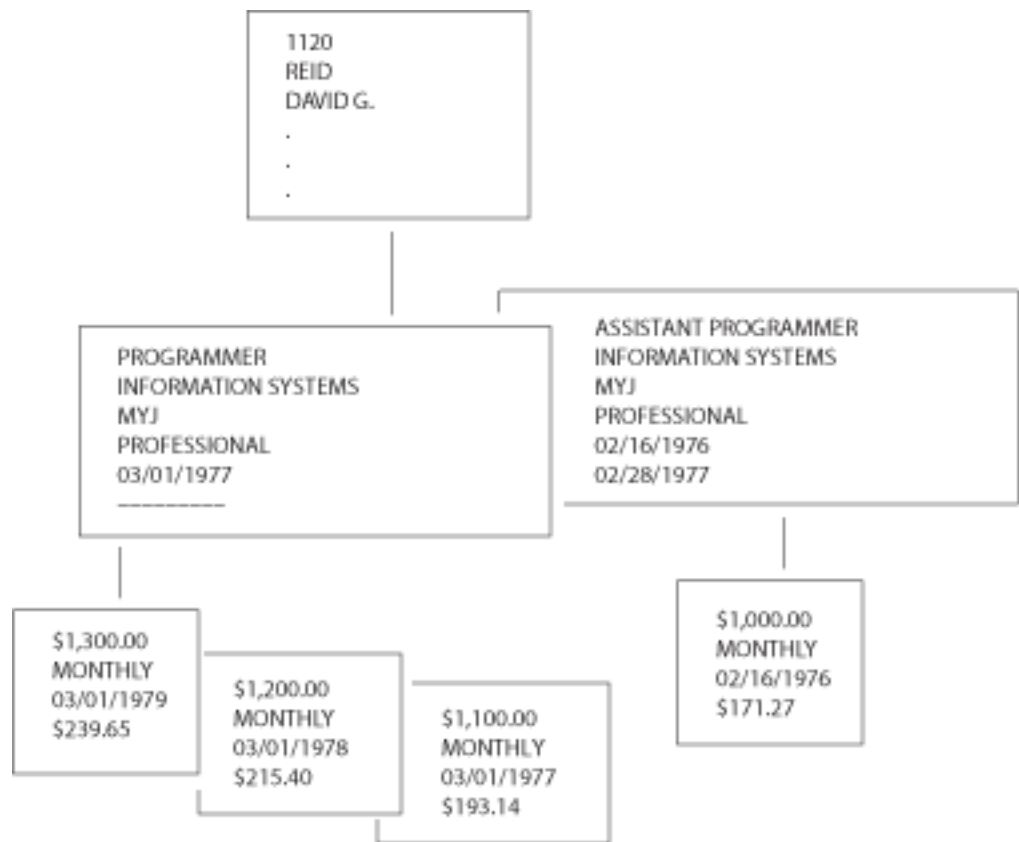
Schema records are *disjoint* if their paths are different. When you create a view descriptor, you cannot include items that are from disjoint schema records. For example, items from C110 and items from C410 cannot be included in the same view descriptor.

Logical Entries

A database consists of groups of logically related data called *logical entries*. The database definition serves as a pattern to create logical entries for the database and to interpret them. A logical entry contains groups of related data called data records. A *data record* is an identifiable set of values that are treated as a unit and associated with a schema record.

For example, in the database Employee, logical entries contain data about employees; all data records that pertain to one employee make up a single logical entry. Each logical entry has a data record for personal data (such as the employee's name, address, and birthday), and a data record that pertains to the position that the employee holds in the company (such as title, department, manager, and pertinent dates). If the employee held several positions in the company, there is a data record for each position.

Using the layout of the database definition, the following figure shows the schema items with values for one employee. David Reid held two positions: programmer and assistant programmer. In addition, he has three salary data records for his programmer position.

Figure 2.3 Logical Entry

When you use a view descriptor in a SAS program to access a SYSTEM 2000 database, you must be familiar with the database structure in order to understand how the interface view engine maps a SYSTEM 2000 logical entry into multiple SAS observations and back again. This process is explained in the next section.

Mapping Data between SAS and SYSTEM 2000

When you create a view descriptor to access data stored in a SYSTEM 2000 database, you define one path in the database. For example, using the database Employee, you can define a view descriptor that includes the items LAST NAME, FORENAME, POSITION TITLE, and PAY RATE. When you access the data using the view descriptor, the interface view engine maps the specified path for each logical entry into multiple observations. The following output shows the logical entry for David Reid mapped into SAS observations.

Output 2.2 Logical Entry Mapped into SAS Observations

LASTNAME	FORENAME	POSITION	PAYRATE
REID	DAVID G.	ASSISTANT PROGRAMMER	\$1,000.00
REID	DAVID G.	PROGRAMMER	\$1,100.00
REID	DAVID G.	PROGRAMMER	\$1,200.00
REID	DAVID G.	PROGRAMMER	\$1,300.00

When browsing SYSTEM 2000 data, such as with the FSVIEW procedure, the results would be similar to those shown in the previous output. (See [Chapter 5, “Browsing and Updating SYSTEM 2000 Data,”](#) on page 41 for more information.)

Null Data (Missing Values)

A logical entry does not need data at every level of the database definition. A logical entry can contain nulls, that is, missing values or empty records.

- A *null item* is a schema item that has no value in the data record. For example, in the logical entry shown in [Figure 2.3 on page 11](#), because David Reid still works for the company, he does not have a value for the schema item END DATE for his programmer position. Therefore, that item is a null item.
- A *null record* is a data record that consists entirely of null items. A null record can occur when there is data for a given data record but no data for its parent record. For example, position information exists but an employee has not been hired yet; there is data at level 1 but the ENTRY record is a null record. Another example is when salary information exists, but position information has not been entered; there is data at level 0 and level 2, but a null record exists at level 1. In both examples, the null record must be present in the database because a parent record must exist for all records except the ENTRY record.
- A *control node* is a schema record that contains no schema items. A control node serves as a control record for descendant records. Look at the database definition for Employee in [Appendix 3, “Example Programs,”](#) on page 137, and you see that schema record 400 is a control node.

Note: In SAS, nulls are referred to as missing values. SYSTEM 2000 and SAS handle nulls (missing values) differently. However, the interface view engine takes care of the differences in a predictable, useful way. See [“Missing Values \(Nulls\)”](#) on page 126 for a discussion of the differences.

SYSTEM 2000 Item Types

Overview of Item Types

Every schema item in a SYSTEM 2000 database has a specified item type. You can specify numeric item types, a date item type, and character item types. The *item type* tells the software how the values for that item are to be stored and displayed. The way you store the values determines how you can use them. For example, values consisting exclusively of digits can be stored in a way that is suitable for computation.

How the values are stored and displayed is also determined by the picture (or length) assigned to an item. For example, a picture for decimal numbers indicates how many digits can be stored and where the decimal point is placed when the values are displayed or used in computation.

When you create a view descriptor, in addition to assigning SAS variable names from the schema item names, PROC ACCESS assigns SAS formats, informats, and lengths using the item's picture. See [“PROC ACCESS Data Conversions”](#) on page 78 for the default SAS variable formats and informats for each SYSTEM 2000 item type.

Numeric Item Types

A numeric item type's picture indicates the number of places required by the longest value expected for an item, and is specified by repeated 9s. For example, 9999 or 9(4) specifies four places. Values for numeric item types cannot exceed their specified picture, that is, overflow is not allowed for numeric values. The following are numeric item types:

INTEGER

stores whole numbers.

DECIMAL

stores numbers with a decimal point.

MONEY

stores numbers with a decimal point, but these values include a floating dollar sign (\$) at the left of the value and CR at the right of the value (if negative) when displayed.

REAL

stores fullword (single-precision), floating-point (or FLOAT) numbers. REAL items do not have a picture. Each REAL value occupies one word (4 bytes) in the database.

DOUBLE

stores double-word (double-precision), floating-point numbers. DOUBLE items do not have a picture. Each DOUBLE value occupies two words (8 bytes). You can also use the DOUBLE item type for storing time values.

Note: SYSTEM 2000 does not have a TIME item type. To store time values, use the DOUBLE item type.

Date Item Types

You can specify date values using the DATE item type. A date does not have an assigned picture.

DATE

stores calendar dates in a fixed format. If the date format is MM/DD/YYYY (the default), the value stored must be in the form 07/04/1989 (including the slashes). You cannot store dates that occurred before the date of origin of the Gregorian calendar, October 15, 1582.

Character Item Types

A character item type's picture corresponds to the number of places that would accommodate most of the values for the item, and is specified by repeated Xs. For example, XXXX or X(4) specifies four places. Values for character item types, except for the UNDEFINED item type, can exceed their picture (up to a maximum of 250 characters) if the specified picture is at least X(4). CHARACTER and TEXT item types have overflow capabilities. The following are character item types:

CHARACTER

stores alphanumeric values with trailing, leading, and extra internal blanks removed. For example, JOHNØØØSMITH is stored and displayed as JOHNØSMITH.

TEXT

stores alphanumeric values, but blanks are not removed. For example, `ØØJOHNØØØSMITHØØØ` is stored and displayed as `ØØJOHNØØØSMITHØØØ`.

UNDEFINED

stores binary bit-string data. UNDEFINED items can contain any of the 256 EBCDIC characters, which are treated like TEXT items except that overflow is not allowed.

Note: When you create a view descriptor, PROC ACCESS assigns default variable lengths to the corresponding SAS variables by using the pictures of the selected items. However, because CHARACTER and TEXT item types have overflow capabilities, there might be values stored in the database that are greater than the default variable length. When you use the view descriptor to select data stored in the database, the larger values are not recognized. Therefore, to access values that exceed their item's picture, you must change the length in the view descriptor definition to the largest possible value stored in the database, up to a maximum of 200.

Therefore, to access values that exceed their item's picture, you must change the length in the view descriptor definition to the largest possible value stored in the database, up to a maximum of 200.

SYSTEM 2000 Indexing

One of the specifications when defining a schema item is whether SYSTEM 2000 is to create an index of its values. SYSTEM 2000 uses the indexes to access the appropriate data records quickly and efficiently.

A schema item for which an index is created is a *key item*, implying that it provides easy access to data records that contain its values. Therefore, a key item has an associated index of every distinct value that occurs for the schema item. However, key values do not have to be unique, and there can be many key items in a database definition, or none.

If a schema item is defined as non-key, its values are not indexed, but the values can be searched sequentially.

In addition, you can create or delete an index of values by using the CREATE INDEX and REMOVE INDEX statements in SYSTEM 2000. Using these statements, SYSTEM 2000 automatically changes the specified item to key or non-key. (For information about these statements, see the SYSTEM 2000 CONTROL Language manual.)

Selecting a Subset of Data

A database would not be very efficient if all logical entries had to be accessed when you needed data from only some of them. SYSTEM 2000 enables you to specify a where-clause to identify those parts of the database that are relevant to your query or update.

A where-clause consists of the keyword WHERE (or WH) and one or more specific conditions that values must meet. Usually, a condition consists of a schema item, an operator, and a value or a range of values. For example:

```
WHERE ACCRUED VACATION EXISTS
WHERE SEX EQ MALE
WHERE BIRTHDAY SPANS 01/01/1949 * 12/31/1949
```

```
WHERE STREET ADDRESS CONTAINS /RIM ROCK/
```

You can also combine conditions by using connector operators to form expressions. For example:

```
WHERE SKILL TYPE = COBOL AND
YEARS OF EXPERIENCE = 4
```

For the SAS/ACCESS interface to SYSTEM 2000, you can include a SYSTEM 2000 where-clause in a view descriptor to specify selection criteria. In addition to or instead of a SYSTEM 2000 where-clause, you can specify selection criteria in a SAS program by using a SAS WHERE clause.

Note: The SYSTEM 2000 where-clause and the WHERE clause in SAS are different. For example, in a SYSTEM 2000 where-clause, the date format (by default) is MM/DD/YYYY, and you do not have to include single quotes around character strings.

For more information, see [“where-clause in SYSTEM 2000” on page 72](#), [“WHERE Clauses in SAS and where-clauses in SYSTEM 2000” on page 127](#), and [“Connecting Strings to Order Conditions” on page 133](#).

Sorting Output

In addition to selecting specific data, SYSTEM 2000 enables you to specify the output order for data through the use of a SYSTEM 2000 ordering-clause, which consists of sort keys that are separated by commas. A *sort key* can be a schema item or a schema record. For each sort key, you can specify whether the output is to be sorted in ascending or descending order. For example, the output produced by the following ordering-clause is first sorted by LAST NAME (C2) in ascending order (the default) and then by YEARS OF EXPERIENCE (C203) in descending order (due to the HIGH specification):

```
OB C2, HIGH C203
```

For the SAS/ACCESS interface to SYSTEM 2000, you can specify data sorting by including a SYSTEM 2000 ordering-clause when you create a view descriptor. In addition, you can specify data sorting in a SAS program using a BY clause.

Note: A BY clause in SAS overrides a SYSTEM 2000 ordering-clause stored in a view descriptor. For more information, see [“ordering-clause in SYSTEM 2000” on page 77](#).

SYSTEM 2000 Passwords

SYSTEM 2000 provides data security with a multi-level password system. Three types of passwords (consisting of 1 to 4 alphanumeric characters with no blanks) can be associated with a SYSTEM 2000 database:

- a master password (required)
- secondary passwords
- a DBA password

The holder of the master password has unqualified access to the database. This is the password under which a database is created. The holder of the master password can also

assign multiple secondary passwords with access authorities assigned to individual database components and a DBA password with access authorities assigned to individual SYSTEM 2000 statements.

The holder of a secondary password can have retrieval (R), update (U), where-clause (W), or no access (N) authority for any combination of database components. An *authority* is a SYSTEM 2000 code that associates a secondary password with a database component and determines what type of access to the database the password allows. For example, the holder of a secondary password can have retrieval and where-clause authority for all database components but no authority to update them.

The DBA password provides a level of security between the master password and secondary passwords. This password enables the DBA to administer databases without being able to access the data stored in them.

For the SAS/ACCESS interface to access SYSTEM 2000 data, you must supply a SYSTEM 2000 password in both the access descriptor and the view descriptor. The passwords can be the same or different. However, the password assigned to the view descriptor must include the data described by the access descriptor. The view descriptor password can be stored in the view, or you can provide (or override) a view descriptor password with a SAS data set option.

SYSTEM 2000 Execution Environments

When you access a SYSTEM 2000 database, you can work in either a single-user or a Multi-User execution environment.

In a single-user environment, you are working with your own copy of SYSTEM 2000 software. You usually have exclusive access to the database. However, the single-user environment can be configured so that all users can query the database.

In a Multi-User environment, many users can access a database at the same time, with queries and updates being handled simultaneously by the Multi-User software for all databases being accessed. The Multi-User environment automatically ensures data protection during concurrent updates, and it automatically guards a database against conflicting tasks.

SYSTEM 2000 Database Files

SYSTEM 2000 has eight database files associated with each database. The first six files are required.

- File 1 - Master Record and Definition Table
- File 2 - Distinct Values Table
- File 3 - Extended Field Table
- File 4 - Multiple Occurrence Table
- File 5 - Hierarchical Table
- File 6 - Data Table
- File 7 - Update Log (optional)
- File 8 - Rollback Log (optional)

When you are working in a single-user environment, you must allocate the appropriate database files in your SAS session before accessing the data. For a Multi-User environment, the database files can be allocated before the Multi-User software is initialized or, in Release 12.0 and later of SYSTEM 2000, the files can be dynamically allocated during execution by using the ALLOC command.

For more information about SYSTEM 2000 terms and concepts, see [Appendix 1, “Topics for Database Administrators,”](#) on page 113 and [Appendix 2, “Advanced Topics for Users,”](#) on page 121.

Chapter 3

SAS/ACCESS Descriptor Files

Overview of SAS/ACCESS Descriptor Files	19
Defining SAS/ACCESS Descriptor Files	19
Creating Descriptor Files	20

Overview of SAS/ACCESS Descriptor Files

SAS interacts with SYSTEM 2000 through the SAS/ACCESS interface view engine, which uses SAS/ACCESS descriptor files. These special files describe the SYSTEM 2000 database and data to SAS.

Defining SAS/ACCESS Descriptor Files

SAS/ACCESS descriptor files are the tools that SAS/ACCESS uses to establish a connection between SAS and SYSTEM 2000. You use the ACCESS procedure to create the two types of descriptor files: access (member type ACCESS) and view (member type VIEW).

An *access descriptor* contains information about the SYSTEM 2000 database that you want to use. The information includes the database name, the item names, and their item types. An access descriptor also contains SAS information, such as the SAS variable names, formats, and informats. Think of an access descriptor as being a *master* descriptor file for one SYSTEM 2000 database because it contains a complete description of that database. You cannot create a single access descriptor that references two SYSTEM 2000 databases. An access descriptor is used to create view descriptors.

A *view descriptor* defines all the data or a subset of the data described by one access descriptor. View descriptor files are sometimes called *SAS views*. This documentation uses *view descriptor* for these files to distinguish them from views created by the SQL procedure.

You choose a subset of data by selecting specific items and specifying selection criteria that the data must meet. For example, you might select the two items LAST NAME and CITY-STATE, and specify that the value stored in item CITY-STATE must be AUSTIN TX. Or, your selection criteria might be the date of transaction and customers' names that begin with W. After you create your view descriptor, you can use it in a SAS program to

read data directly from the SYSTEM 2000 database or to extract the data and place it in a SAS data file. You can also specify a sequence order for the data.

For each access descriptor that you define, you usually have several view descriptors. Each of these view descriptors contains different subsets of data.

Creating Descriptor Files

Access and view descriptor files are created by using the ACCESS procedure. You can create these files by using one PROC ACCESS step or multiple separate PROC ACCESS steps. This section shows how to create descriptor files in one PROC ACCESS step. Within a step, you can define multiple descriptor files of the same type or of different types.

Examples for creating the access descriptor MyLib.Employe and the view descriptors Vlib.EmpPos and Vlib.EmpSkil by executing separate PROC ACCESS steps are provided in [Appendix 3, “Example Programs,” on page 137](#).

Note: When you execute a separate PROC ACCESS step to create a view descriptor, you must use the ACCDESC= option to specify an existing access descriptor from which the view descriptor is created.

The most common way to use the PROC ACCESS statements, especially when using batch mode, is to create an access descriptor and one or more view descriptors based on this access descriptor in a single execution of PROC ACCESS. For example, in the program that follows, first, you create the access descriptor MyLib.Employe. Then, you create the two view descriptors Vlib.EmpPos and Vlib.EmpSkil. In the section that immediately follows this example program, each statement is explained in the order in which it appears in the program.

```
proc access dbms=s2k;
  create mylib.employe.access;
    database=employee;
    s2kpw=demo mode=multi;
    assign=yes;
    drop c110 c120;
    rename forename=firstnme office_e=phone
           yearsofe=years gender=sex
           degree_c=degree;
    length firstnme=13 lastname=13 c101=16;
    list all;

  create vlib.emppos.view;
    select lastname firstnme position departme manager;
    subset "order by lastname";
    list all;

  create vlib.empskil.view;
    select c2 c3 c201 c203;
    subset "ob skilltyp";
    s2kpw=demo mode=multi;
    list view;
run;
```

proc access dbms=s2k;

invokes the ACCESS procedure for the SAS/ACCESS interface to SYSTEM 2000.

create mylib.employe.access;
 identifies the access descriptor, MyLib.Employe, that you want to create. The libref MyLib must be associated with the SAS library before you can specify it in the CREATE statement.

database=employee
 indicates that this access descriptor is for the database Employee.

s2kpw=demo mode=multi;
 specifies the password DEMO (which is required to access the database definition), and indicates that the database is in the Multi-User environment.

assign=yes;
 generates unique SAS variable names based on the first 8 non-blank characters of the item names. Variable names and attributes can be changed in this access descriptor but not in any view descriptors that are created from this access descriptor.

drop c110 c120;
 marks the records associated with the C-numbers C110 and C120 as non-display. Because these C-numbers represent records, all the items in each record are marked as non-display. Therefore, none of the items in the two records associated with these numbers appear in any view descriptor created from this access descriptor.

**rename forename=firstnme office_e=phone yearsofe=years
 gender=sex degree_c=degree;**
 renames the default SAS variable names associated with the SAS names FORENAME, OFFICE_E, YEARSOFE, GENDER, and DEGREE_C. You specify the default SAS variable name on the left side of the equal sign (=) and the new name on the right side of the equal sign. Because the ASSIGN=YES statement was specified earlier, any view descriptors created from this access descriptor automatically use the new SAS variable names.

length firstnme=13 lastname=13 c101=16;
 changes the field width for the items associated with FIRSTNME and LASTNAME to 13 characters and the field width for the item associated with C-number C101 (the SAS name POSITION) to 16 characters.

list all;
 lists the access descriptor's item identifier numbers, C-numbers, SAS variable names, SAS formats, SAS informats, and SAS variable lengths. The list also includes any associated information specified in the BYKEY statement. Items that have been dropped from display (by using the DROP statement) have *NON-DISPLAY* next to them. The list is written to the SAS log.

create vlib.emppos.view;
 writes the access descriptor to the library associated with MyLib and identifies the view descriptor, Vlib.EmpPos, that you want to create. The libref Vlib must be associated with a SAS library before you can specify it in this statement.

select lastname firstnme position departme manager;
 selects the items associated with the SAS names LASTNAME, FIRSTNME, POSITION, DEPARTME, and MANAGER for inclusion in the view descriptor. The SELECT statement is required to create the view unless a RENAME, FORMAT, INFORMAT, LENGTH, or BYKEY statement is specified.

subset "order by lastname";
 specifies that you want SYSTEM 2000 to order (or sort) output data set by last name. Use SYSTEM 2000 syntax in the SUBSET statement. For more information, see the *SYSTEM 2000 Software: QUEST Language and System-Wide Commands, Version 12, First Edition* manual.

```
list all;
```

lists all the available item identifier numbers, C-numbers, SAS variable names, SAS formats, SAS informats, and SAS variable lengths on which the view descriptor is based. The list also includes any associated information specified in a BYKEY statement and selection criteria specified in the view descriptor. Items that have been dropped from the display have *NON-DISPLAY* next to them. Items that have been selected for the view have *SELECTED* next to them. The list is written to the SAS log.

```
create vlib.empskil.view;
```

writes the first view descriptor to the library associated with Vlib and identifies the next view descriptor, Vlib.EmpSkil, that you create in this example.

```
select c2 c3 c201 c203;
```

selects the four items associated with the C-numbers C2, C3, C201, and C203 for inclusion in the view descriptor. The SELECT statement is required to create the view unless a RENAME, FORMAT, INFORMAT, LENGTH, or BYKEY statement is specified.

```
subset "ob skilltyp";
```

specifies that you want the observations to be sorted by skill type. See [“SUBSET Statement \(Optional\)” on page 88](#) for syntax information.

```
s2kpw=demo mode=multi;
```

specifies the password required to access the data and indicates the database is in the Multi-User environment. This information is stored in the view descriptor. To override this password or to specify a SYSTEM 2000 password for the view descriptor Vlib.EmpPos that omits the S2KPW statement, you can use the S2KPW data set option. For more information, see [“Overriding Options” on page 121](#).

```
list view;
```

lists the item identifier numbers, the C-numbers, the SAS variable names, the SAS formats, the SAS informats, and the SAS variable lengths that have been selected for the view descriptor. The list also includes any associated information specified in a BYKEY statement and selection criteria specified in the view descriptor. The list is written to the SAS log.

```
run;
```

writes the last view descriptor and runs the program.

Chapter 4

SYSTEM 2000 Data in SAS Programs

Using SYSTEM 2000 Data in SAS	23
Reviewing Variables	24
Printing Data	25
Charting Data	26
Calculating Statistics	27
Selecting and Combining Data with the SQL Procedure	30
Using the WHERE Clause	30
Combining Data from Various Sources	31
Creating New Items with the GROUP BY Clause in PROC SQL	34
Updating SAS Data Files with SYSTEM 2000 Data	35
Using the UPDATE Statement	35
Updating Data Files in SAS 7 and Later	37
Performance Considerations	38

Using SYSTEM 2000 Data in SAS

One advantage of the SAS/ACCESS interface to SYSTEM 2000 is that it enables SAS to read and write SYSTEM 2000 data directly, using SAS programs. This section presents examples of using SYSTEM 2000 data that is described by view descriptors in SAS programs. For information about the example data, see [Appendix 3, “Example Programs,” on page 137](#). For information about using view descriptors efficiently in SAS programs, see [“Performance Considerations” on page 38](#).

Throughout the examples, the SAS terms *variable* and *observation* are used in place of comparable SYSTEM 2000 terms because these examples illustrate using SAS procedures and the DATA step. The examples also include printing and charting data, using the SQL procedure to combine data from various sources, and updating a SAS 7 data set with data from SYSTEM 2000. For more information about the SAS language and procedures used in the examples, see the documents referred to throughout this section.

Reviewing Variables

Suppose that, in your SAS program that you want to use SYSTEM 2000 data that is described by a view descriptor, but you cannot remember the variable names or formats and informats. You can get this information by using the CONTENTS or the DATASETS procedure.

The following example uses PROC DATASETS to give you information about the view descriptor Vlib.EmpPos, which you created earlier. See [Chapter 3, “SAS/ACCESS Descriptor Files,”](#) on page 19. The following output shows the results.

```
proc datasets library=vlib memtype=view;
  contents data=emppos(s2kpw=demo);
run;
```

Output 4.1 DATASETS Procedure Results with a View Descriptor

The SAS System								1
DATASETS PROCEDURE								
Data Set Name:	VLIB.EMPPOS	Observations:						887
Member Type:	VIEW	Variables:						5
Engine:	SASIOS2K	Indexes:						0
Created:	03NOV89:16:17:59	Observation Length:						53
Last Modified:	07SEP89:14:15:58	Deleted Observations:						0
Data Set Type:		Compressed:						NO
Label:								
-----Alphabetic List of Variables and Attributes-----								
#	Variable	Type	Len	Pos	Format	Informat	Label	
4	DEPARTME	Char	14	36	\$14.	\$14.	DEPARTMENT	
2	FIRSTNME	Char	10	10	\$10.	\$10.	FORENAME	
1	LASTNAME	Char	10	0	\$10.	\$10.	LAST NAME	
5	MANAGER	Char	3	50	\$3.	\$3.	MANAGER	
3	POSITION	Char	16	20	\$16.	\$16.	POSITION	
TITLE								

Notice the following in the preceding output:

- Because you cannot change a view descriptor's variable labels when you use PROC DATASETS, the labels that are generated are the complete SYSTEM 2000 item names at the time that the view descriptor was created. The labels cannot be overridden.
- The Created date is the date when the access descriptor for this view descriptor was created.
- The Last Modified date is the last date the SYSTEM 2000 database was updated.
- The Observations number is the highest number of schema records that occurred in the database. The number of observations shown here does not correspond to the number of observations that the view descriptor accesses.

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

Printing Data

You can use the PRINT procedure to print SYSTEM 2000 data that is described by view descriptors in the same way that you use PROC PRINT with SAS data files. See the output, which shows the first page of output produced by the following program:

```
proc print data=vlib.emppos(s2kpw=demo);
    title2 'Subset of EMPLOYEE Database Information';
run;
```

Output 4.2 Results of PROC PRINT for SYSTEM 2000 Data

Subset of EMPLOYEE Database Information						1
OBS	LASTNAME	FIRSTNME	POSITION	DEPARTME	MANAGER	
1			PROGRAMMER	INFORMATION SY	MYJ	
2	AMEER	DAVID	SR SALES REPRESE	MARKETING	VPB	
3	AMEER	DAVID	JR SALES REPRESE	MARKETING	VPB	
4	BOWMAN	HUGH E.	EXECUTIVE VICE-P	CORPORATION	CPW	
5	BROOKS	RUBEN R.	JR SALES REPRESE	MARKETING	MAS	
6	BROWN	VIRGINA P.	MANAGER WESTERN	MARKETING	OMG	
7	CAHILL	JACOB	MANAGER SYSTEMS	INFORMATION SY	JBM	
8	CANADY	FRANK A.	MANAGER PERSONNE	ADMINISTRATION	PRK	
9	CHAN	TAI	SR SALES REPRESE	MARKETING	TZR	
10	COLLINS	LILLIAN	MAIL CLERK	ADMINISTRATION	SQT	
11	FAULKNER	CARRIE ANN	SECRETARY	CORPORATION	JBM	
12	FERNANDEZ	SOPHIA	STANDARDS & PROC	INFORMATION SY	JLH	
13	FREEMAN	LEOPOLD	SR SYSTEMS PROGR	INFORMATION SY		
JLH						

When you use PROC PRINT, you might want to use the OBS= option, which enables you to specify the last observation to be processed. This is especially useful when the view descriptor describes large amounts of data, or when you just want to see a sample of the output. The following program uses the OBS= option to print the first five rows described by the view descriptor Vlib.EmpPos. The results follow the program.

```
proc print data=vlib.emppos(s2kpw=demo obs=5);
    title2 'First Five Data Rows Described by VLIB.EMPPPOS';
run;
```

Output 4.3 Results of Using the OBS= Option in PROC PRINT

First Five Data Rows Described by VLIB.EMPPOS						1
OBS	LASTNAME	FIRSTNME	POSITION	DEPARTME	MANAGER	
1			PROGRAMMER	INFORMATION SY	MYJ	
2	AMEER	DAVID	SR SALES REPRESE	MARKETING	VPB	
3	AMEER	DAVID	JR SALES REPRESE	MARKETING	VPB	
4	BOWMAN	HUGH E.	EXECUTIVE VICE-P	CORPORATION	CPW	
5	BROOKS	RUBEN R.	JR SALES REPRESE	MARKETING		
MAS						

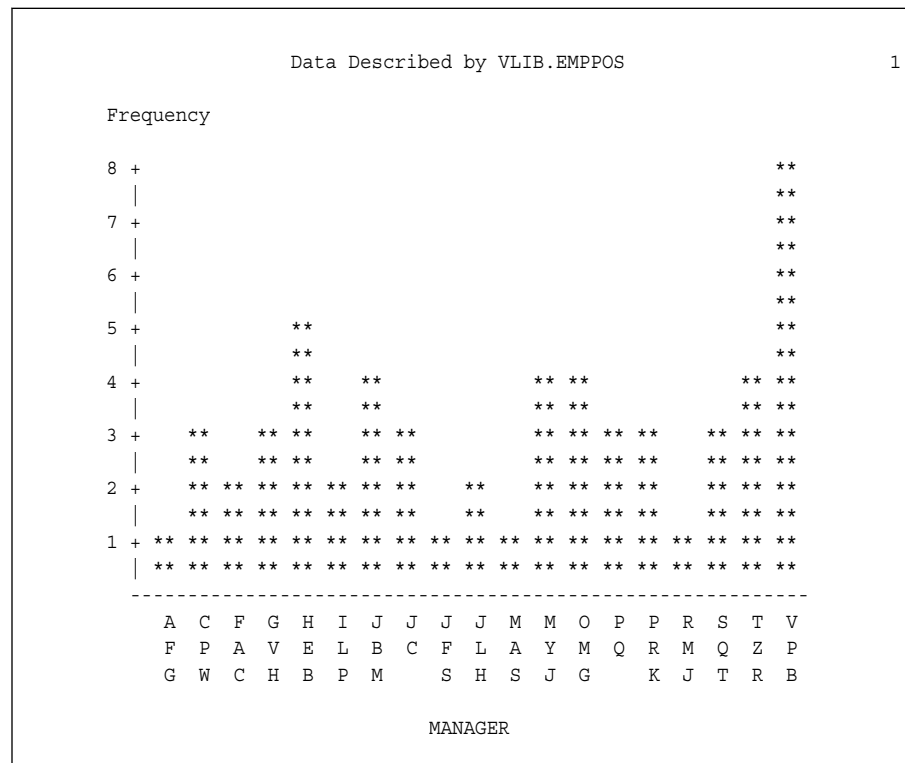
The FIRSTOBS= option can also be used with view descriptors. However, the FIRSTOBS= option does not improve performance significantly because each record must be read and its position calculated.

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

Charting Data

You can use the CHART procedure to chart data that is described by view descriptors in the same way that you use PROC CHART with SAS data files. See the output, which shows the output produced by the following program, that uses the view descriptor Vlib.EmpPos to create a vertical bar chart of the number of employees each manager has had. The number of employees for each manager is represented by the height of the bar.

```
proc chart data=vlib.emppos(s2kpw=demo);
  vbar manager;
  title2 'Data Described by VLIB.EMPPOS';
run;
```

Output 4.4 Vertical Bar Chart Showing Number of Employees per Manager

For more information about the CHART procedure, see the *Base SAS Procedures Guide*. If you have SAS/GRAPH software licensed at your site, you can create colored block charts, plots, and other graphics based on SYSTEM 2000 data. See *SAS/GRAPH: Reference* for more information.

Calculating Statistics

The statistical procedures FREQ, MEANS, and RANK can be used with SYSTEM 2000 data.

The following program uses PROC FREQ to calculate the percentage of employees that have each of the college degrees that exist in the database Employee. This program uses the view descriptor Vlib.EmpEduc. The output shows the results.

```
proc freq data=vlib.empeduc;
  tables degree;
  title2 'Data Described by VLIB.EMPEDUC';
run;
```

Output 4.5 One-Way Frequency Table for Item DEGREE in View Descriptor Vlib.EmpEduc

Data Described by VLIB.EMPEDUC					1
DEGREE/CERTIFICATE					
DEGREE	Frequency	Percent	Cumulative Frequency	Cumulative Percent	

AA	5	7.9	5	7.9	
BA	12	19.0	17	27.0	
BS	23	36.5	40	63.5	
HIGH SC	6	9.5	46	73.0	
MA	3	4.8	49	77.8	
MBA	1	1.6	50	79.4	
MS	9	14.3	59	93.7	
PHD	4	6.3	63	100.0	
Frequency Missing = 12					

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

In a further analysis of employee background, suppose you also want to create some statistics about skill types of the employees and their years of experience. The view descriptor Vlib.EmpSkil accesses the values from the database Employee, and the following program uses PROC MEANS to generate the mean and sum of the years of experience by skill type. The number of observations (N) and the number of missing values (NMISS) are also included.

Notice that the BY statement causes the interface view engine to generate a SYSTEM 2000 ordering-clause so that the data is sorted by skill type. The output shows some of the results produced by this program.

```
proc means data=vlib.empskil mean sum n nmiss
  maxdec=0;
  by skilltyp;
  var years;
  title2 'Data Described by VLIB.EMPSKIL';
run;
```


Output 4.6 Statistics for Skill Type and Years of Experience

Data Described by VLIB.EMPSKIL				1
Analysis Variable : YEARS YEARS OF EXPERIENCE				
----- SKILL TYPE= -----				
N	Nmiss	Mean	Sum	
0	6	.	.	
----- SKILL TYPE=ACCOUNTING -----				
N	Nmiss	Mean	Sum	
6	0	8	47	
----- SKILL TYPE=ASSEMBLER -----				
N	Nmiss	Mean	Sum	
14	0	10	141	
----- SKILL TYPE=CARTOON ART -----				
N	Nmiss	Mean	Sum	
1	0	1	1	
----- SKILL TYPE=CHINESE -----				
N	Nmiss	Mean	Sum	
1	0	8	8	
----- SKILL TYPE=COBOL -----				
N	Nmiss	Mean	Sum	
12	0	7	88	

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

You can also use more advanced statistics procedures with SYSTEM 2000 data. The following program uses PROC RANK with data described by the view descriptor Vlib.EmpBd to calculate the order of birthdays for a group of employees, and to assign

the variable name DATERNK to the new item created by PROC RANK. (The Vlib.EmpBd view descriptor includes a SYSTEM 2000 where-clause to select only the employees in the Marketing Department.) The output shows some of the results from this program.

```
proc rank data=vlib.empbd out=mydata.rankexm;
  var birthday;
  ranks daternk;
run;

proc print data=mydata.rankexm;
  title2 'Order of Marketing Employee Birthdays';
run;
```

Output 4.7 Ranking of Employee Birthdays

Order of Marketing Employee Birthdays					1
OBS	LASTNAME	FIRSTNME	BIRTHDAY	DATERNK	
1	AMEER	DAVID	10OCT51	14.0	
2	BROOKS	RUBEN R.	25FEB52	15.0	
3	BROWN	VIRGINA P.	24MAY46	9.0	
4	CHAN	TAI	04JUL46	10.0	
5	GARRETT	OLAN M.	23JAN35	2.0	
6	GIBSON	GEORGE J.	23APR46	8.0	
7	GOODSON	ALAN F.	21JUN50	13.0	
8	JUAREZ	ARMANDO	28MAY47	11.0	
9	LITTLEJOHN	FANNIE	17MAY54	17.0	
10	RICHARDSON	TRAVIS Z.	30NOV37	4.0	
11	RODRIGUEZ	ROMUALDO R.	09FEB29	1.0	
12	SCHOLL	MADISON A.	19MAR45	7.0	
13	SHROPSHIRE	LELAND G.	04SEP49	12.0	
14	SMITH	JERRY LEE	13SEP42	5.5	
15	VAN HOTTEN	GWENDOLYN	13SEP42	5.5	
16	WAGGONNER	MERRILEE D.	27APR36	3.0	
17	WILLIAMSON	JANICE L.	19MAY52	16.0	

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

Selecting and Combining Data with the SQL Procedure

Using the WHERE Clause

Suppose you have two view descriptors, Vlib.EmpPos and Vlib.EmpEduc, that access employee positions and employee education, respectively. You can use PROC SQL to combine these files into a single SAS data file. The WHERE clause in SAS specifies that you want a data file that contains information about employees for whom the value for their level of education is missing, and who are in the department CORPORATION.

Note: PROC SQL displays the variable labels as stored in the view. However, because you are referencing a view descriptor, you must use the SAS variable names in the WHERE clause, not the SYSTEM 2000 item names.

The output shows the results from this example. (Notice that Waterhouse appears twice in the output. This is because he has two values for schema item C411 SCHOOL, but neither value has an associated value for C412 DEGREE/CERTIFICATE.)

```
proc sql;
  title 'Corporation Positions With No Degrees';
  select emppos.lastname, position, degree, departme
  from vlib.emppos, vlib.empeduc
  where emppos.lastname=empeduc.lastname and
        empeduc.degree is missing and
        emppos.departme='CORPORATION'
  order by lastname;
```

Output 4.8 Output from SQL Procedure with a WHERE Clause

Corporation Positions With No Degrees				1
LAST NAME	POSITION TITLE	DEGREE/CERTIFICATE	DEPARTMENT	
FAULKNER	SECRETARY		CORPORATION	
KNIGHT	SECRETARY		CORPORATION	
WATERHOUSE	PRESIDENT		CORPORATION	
WATERHOUSE	PRESIDENT		CORPORATION	

Combining Data from Various Sources

Suppose that, along with the view descriptors Vlib.EmpPos and Vlib.EmpEduc, you have the SAS data file MyData.Classes that contains in-house continuing education classes taken by employees. You can use PROC SQL to join these sources of data to form a single output table of employee names, their departments, their degrees, and the in-house classes that they have taken. The output (Data Described by the View Descriptor Vlib.EmpPos, Data Described by the View Descriptor Vlib.EmpEduc, and SAS Data File MyData.Classes) shows the results of using PROC PRINT on the data described by Vlib.EmpPos and Vlib.EmpEduc and in the file MyData.Classes.

```
proc print data=vlib.emppos;
  title2 'Data Described by VLIB.EMPPOS';
run;

proc print data=vlib.empeduc;
  title2 'Data Described by VLIB.EMPEDUC';
run;

proc print data=mydata.classes;
  title2 'SAS Data File MYDATA.CLASSES';
run;
```

Note: If you have many PROC SQL views and view descriptors, you might want to store the PROC SQL views in a separate SAS library from your view descriptors. They both have the member type VIEW, so you cannot tell a view descriptor from a PROC SQL view.

Output 4.9 Data Described by the View Descriptor *Vlib.EmpPos*

Data Described by VLIB.EMPPOS						1
OBS	LASTNAME	FIRSTNME	POSITION	DEPARTME	MANAGER	
1			PROGRAMMER	INFORMATION SY	MYJ	
2	AMEER	DAVID	SR SALES REPRESE	MARKETING	VPB	
3	AMEER	DAVID	JR SALES REPRESE	MARKETING	VPB	
4	BOWMAN	HUGH E.	EXECUTIVE VICE-P	CORPORATION	CPW	
5	BROOKS	RUBEN R.	JR SALES REPRESE	MARKETING	MAS	
6	BROWN	VIRGINA P.	MANAGER WESTERN	MARKETING	OMG	
7	CAHILL	JACOB	MANAGER SYSTEMS	INFORMATION SY	JBM	
8	CANADY	FRANK A.	MANAGER PERSONNE	ADMINISTRATION	PRK	
9	CHAN	TAI	SR SALES REPRESE	MARKETING	TZR	
10	COLLINS	LILLIAN	MAIL CLERK	ADMINISTRATION	SQT	
11	FAULKNER	CARRIE ANN	SECRETARY	CORPORATION	JBM	
12	FERNANDEZ	SOPHIA	STANDARDS & PROC	INFORMATION SY	JLH	
13	FREEMAN	LEOPOLD	SR SYSTEMS PROGR	INFORMATION SY	JLH	

Output 4.10 Data Described by the View Descriptor *Vlib.EmpEduc*

Data Described by VLIB.EMPEDUC					1
OBS	LASTNAME	FIRSTNME	SEX	DEGREE	
1					
2	AMEER	DAVID	MALE	BS	
3	BOWMAN	HUGH E.	MALE	MS	
4	BOWMAN	HUGH E.	MALE	BS	
5	BOWMAN	HUGH E.	MALE	PHD	
6	BROOKS	RUBEN R.	MALE	BS	
7	BROWN	VIRGINA P	FEMALE	BA	
8	CAHILL	JACOB	MALE	BS	
9	CAHILL	JACOB	MALE	BS	
10	CANADY	FRANK A.	MALE	MA	
11	CANADY	FRANK A.	MALE	BS	
12	CHAN	TAI	MALE	PHD	
13	CHAN	TAI	MALE	BA	

Output 4.11 SAS Data File *MyData.Classes*

SAS Data File MYDATA.CLASSES				1
OBS	LASTNAME	FIRSTNME	CLASS	
1	AMEER	DAVID	PRESENTING IDEAS	
2	CANADY	FRANK A.	PRESENTING IDEAS	
3	GIBSON	MOLLY I.	SUPERVISOR SKILLS	
4	GIBSON	MOLLY I.	STRESS MGMT	
5	RICHARDSON	TRAVIS Z.	SUPERVISOR SKILLS	

The following SAS program selects and combines data from these three sources (the two view descriptors and the SAS data file) to create the view *SQL.EDUC*. This view retrieves employee names, their departments, their levels of education, and the in-house classes that they have taken.

In the following program, the CREATE VIEW statement incorporates a WHERE clause as part of the SELECT statement. The last SELECT statement retrieves and displays the PROC SQL view SQL.EDUC. To select all items from the view, an asterisk (*) is used in place of item names. The order of the items that are displayed matches the order of the items as they are specified in the first SELECT clause.

The output following the program shows the data described by the SQL.EDUC view. PROC SQL uses variable labels in the output by default.

```
proc sql;
  create view sql.educ as
  select emppos.lastname, emppos.firstnme,
         emppos.departme, empeduc.degree,
         classes.class as course
  from vlib.emppos,
       vlib.empeduc,
       mydata.classes
  where (emppos.lastname=empeduc.lastname
        and emppos.firstnme=empeduc.firstnme)
        and
        (empeduc.lastname=classes.lastname
        and empeduc.firstnme=classes.firstnme)
  order by emppos.lastname, course;

  title 'Data Described by SQL.EDUC';
  select * from sql.educ;
```

Output 4.12 Data Described by the PROC SQL View SQL.EDUC

Data Described by SQL.EDUC				1
LAST NAME	FORENAME	DEPARTMENT	DEGREE/CERTIFICATE COURSE	
AMEER	DAVID	MARKETING	BS	
PRESENTING IDEAS				
AMEER	DAVID	MARKETING	BS	
PRESENTING IDEAS				
CANADY	FRANK A.	ADMINISTRATION	MA	
PRESENTING IDEAS				
CANADY	FRANK A.	ADMINISTRATION	BS	
PRESENTING IDEAS				
GIBSON	MOLLY I.	INFORMATION SY	BA	
STRESS MGMT				
GIBSON	MOLLY I.	INFORMATION SY	BA	
SUPERVISOR SKILLS				
RICHARDSON	TRAVIS Z.	MARKETING	BS	
SUPERVISOR SKILLS				

The view SQL.EDUC lists entries for employees who have taken in-house classes, their departments, and their degrees. However, it contains duplicate observations because some employees have more than one degree and have taken more than one in-house class. To make the data more readable, you can create the final SAS data file

MyData.Update by using the SET statement and the special variable FIRST. This variable identifies which observation is the first in a specific BY group. You need each employee's name associated only one time with the degrees and in-house education classes that were taken, regardless of the number of degrees or the number of classes taken.

The output that follows the program displays the data file MyData.Update that contains an observation for each unique combination of employee, degree, and in-house class.

```
data mydata.update;
  set sql.educ;
  by lastname course;
  if first.lastname then output;
run;

proc print;
  title2 'MYDATA.UPDATE Data File';
run;
```

Output 4.13 SAS Data File MyData.Update

MYDATA.UPDATE Data File						1
OBS	LASTNAME	FIRSTNME	DEPARTME	DEGREE	COURSE	
1	AMEER	DAVID	MARKETING	BS	PRESENTING IDEAS	
2	CANADY	FRANK A.	ADMINISTRATION	MA	PRESENTING IDEAS	
3	GIBSON	MOLLY I.	INFORMATION SY	BA	STRESS MGMT	
4	RICHARDSON	TRAVIS Z.	MARKETING	BS	SUPERVISOR	
SKILLS						

Creating New Items with the GROUP BY Clause in PROC SQL

It is often useful to create new items with summary or aggregate functions such as AVG or SUM. You can easily use PROC SQL with data described by a view descriptor to display output that contains new items.

The following program uses PROC SQL to retrieve and manipulate data from the view descriptor Vlib.EmpVac. When this query (as a SELECT statement is often called) is submitted, it calculates and displays the average vacation time (in hours) for each department. The order of the items displayed matches the order of the items as specified in the SELECT clause of the query. The output for follows the program shows the results from using the SELECT statement.

```
proc sql;
  title 'Average Vacation Per Department';
  select distinct departme,
    avg(accruedv) label='Avg Vac'
  from vlib.empvac
  where departme is not missing
  group by departme;
```

Output 4.14 Data Retrieved by a PROC SQL Query

Average Vacation Per Department	
DEPARTMENT	Avg
ADMINISTRATION	43
CORPORATION	40.72727
INFORMATION SY	61.75
MARKETING	47.61905

For more information about the SQL procedure, see the *SAS SQL Procedure User's Guide*.

Updating SAS Data Files with SYSTEM 2000 Data

Using the UPDATE Statement

You can update a SAS data file with SYSTEM 2000 data that is described by a view descriptor just as you can update a SAS data file by using another data file, that is, by using an UPDATE statement in a DATA step. In this section, the term *transaction data* refers to the new data that is added to the original file. Because the SAS/ACCESS interface to SYSTEM 2000 uses the SAS 6 compatibility engine, the transaction data is from a SAS 6 source. However, the original file can be a SAS 6 or later data file.

Suppose you have the SAS 6 data file V6.Birthdy that contains the names and birthdays of the employees in Marketing. The file is out-of-date, and you want to update it with data described by Vlib.EMPBD. To perform the update, submit the following program:

```
proc sort data=v6.birthdy;
  by lastname;
run;

data mydata.newbday;
  update v6.birthdy vlib.empbd;
  by lastname firstnme;
run;
```

In this example, when the UPDATE statement references the view descriptor Vlib.EMPBD and uses a BY statement in the DATA step, the BY statement causes the interface view engine to automatically generate a SYSTEM 2000 ordering-clause for the variable LASTNAME. The ordering-clause causes the SYSTEM 2000 data to be presented to SAS already sorted so that the SYSTEM 2000 DATA can be used to update the data file MyData.NewBday. The data file V6.Birthdy had to be sorted before the update because the UPDATE statement needs the data sorted by the BY variable.

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

Output 4.15 Data File to Be Updated, V6.Birthdy

V6.BIRTHDY Data File				1
OBS	LASTNAME	FIRSTNME	BIRTHDAY	
1	JONES	FRANK	22MAY53	
2	MCVADE	CURTIS	25DEC54	
3	SMITH	VIRGINIA	14NOV49	
4	TURNER	BECKY	26APR50	

Output 4.16 Data Described by the View Descriptor Vlib.EmpBd

Data Described by VLIB.EMPBD				1
OBS	LASTNAME	FIRSTNME	BIRTHDAY	
1	AMEER	DAVID	10OCT51	
2	BROOKS	RUBEN R.	25FEB52	
3	BROWN	VIRGINA P.	24MAY46	
4	CHAN	TAI	04JUL46	
5	GARRETT	OLAN M.	23JAN35	
6	GIBSON	GEORGE J.	23APR46	
7	GOODSON	ALAN F.	21JUN50	
8	JUAREZ	ARMANDO	28MAY47	
9	LITTLEJOHN	FANNIE	17MAY54	
10	RICHARDSON	TRAVIS Z.	30NOV37	
11	RODRIGUEZ	ROMUALDO R	09FEB29	
12	SCHOLL	MADISON A.	19MAR45	
13	SHROPSHIRE	LELAND G.	04SEP49	
14	SMITH	JERRY LEE	13SEP42	
15	VAN HOTTEN	GWENDOLYN	13SEP42	
16	WAGGONNER	MERRILEE D	27APR36	
17	WILLIAMSON	JANICE L.	19MAY52	

Output 4.17 Updated Data File, MyData.NewBday

MYDATA.NEWBDAY Data File				1
OBS	LASTNAME	FIRSTNME	BIRTHDAY	
1	AMEER	DAVID	10OCT51	
2	BROOKS	RUBEN R.	25FEB52	
3	BROWN	VIRGINA P.	24MAY46	
4	CHAN	TAI	04JUL46	
5	GARRETT	OLAN M.	23JAN35	
6	GIBSON	GEORGE J.	23APR46	
7	GOODSON	ALAN F.	21JUN50	
8	JONES	FRANK	22MAY53	
9	JUAREZ	ARMANDO	28MAY47	
10	LITTLEJOHN	FANNIE	17MAY54	
11	MCVADE	CURTIS	25DEC54	
12	RICHARDSON	TRAVIS Z.	30NOV37	
13	RODRIGUEZ	ROMUALDO R	09FEB29	
14	SCHOLL	MADISON A.	19MAR45	
15	SHROPSHIRE	LELAND G.	04SEP49	
16	SMITH	JERRY LEE	13SEP42	
17	SMITH	VIRGINIA	14NOV49	
18	TURNER	BECKY	26APR50	
19	VAN HOTTEN	GWENDOLYN	13SEP42	
20	WAGGONNER	MERRILEE D	27APR36	
21	WILLIAMSON	JANICE L.	19MAY52	

Updating Data Files in SAS 7 and Later

Beginning with SAS 7, SAS supports different naming conventions than those used in SAS 6. Therefore, there might be character-length discrepancies between the variables in an original data file and the transaction data. You have two choices when updating a SAS 7 and later data file with the data described by a view descriptor:

- let the compatibility engine truncate names that exceed eight characters. The truncated variables are added to the updated data file as new variables.
- rename the variables in the data file in SAS 7 and later to match the variable names in the descriptor file.

The following program resolves character-length discrepancies by using the RENAME option in the UPDATE statement in the DATA step. The SAS 7 data file V7.Consulting_Birthdays, which contains Consulting names and birthdays, is updated with data described by Vlib.EmpBd. In this program, the updated SAS data file NewData.New_Birthdays is a SAS 7 data file stored in the SAS 7 SAS library associated with the libref NewData. The RENAME= option in the DATA step is used in the UPDATE statement to rename the variables before the updated data file NewData.New_Birthdays is created. The two outputs that follow the program show the results of PROC PRINT on the original data file and the updated data file.

```
proc sort data=v7.consulting_birthdays;
  by last_name;
run;

data newdata.new_birthdays;
  update v7.consulting_birthdays
  (rename=(last_name=lastname
           first_name=firstnme
```

```

        birthdate=birthday)) vlib.empbd;
    by lastname firstnme;
run;

```

Output 4.18 Data File to Be Updated, V7.Consulting_Birthdays,

V7.Consulting_Birthdays Data File				1
obs	last_name	first_name	birthdate	
1	JOHNSON	ED	30JAN65	
2	LEWIS	THOMAS	25MAY54	
3	SMITH	AMANDA	02DEC60	
4	WILSON	REBECCA	13APR58	

Output 4.19 Updated Data File, V7.New_Birthdays

V7.NEW_BIRTHDAYS Data File				1
obs	lastname	firstnme	birthday	
1	AMEER	DAVID	10OCT51	
2	BROOKS	RUBEN R.	25FEB52	
3	BROWN	VIRGINA P.	24MAY46	
4	CHAN	TAI	04JUL46	
5	GARRETT	OLAN M.	23JAN35	
6	GIBSON	GEORGE J.	23APR46	
7	GOODSON	ALAN F.	21JUN50	
8	JOHNSON	ED	30JAN65	
9	JUAREZ	ARMANDO	28MAY47	
10	LEWIS	THOMAS	25MAY54	
11	LITTLEJOHN	FANNIE	17MAY54	
12	RICHARDSON	TRAVIS Z.	30NOV37	
13	RODRIGUEZ	ROMUALDO R	09FEB29	
14	SCHOLL	MADISON A.	19MAR45	
15	SHROPSHIRE	LELAND G.	04SEP49	
16	SMITH	AMANDA	02DEC60	
17	SMITH	JERRY LEE	13SEP42	
18	VAN HOTTEN	GWENDOLYN	13SEP42	
19	WAGGONNER	MERRILEE D	27APR36	
20	WILLIAMSON	JANICE L.	19MAY52	
21	WILSON	REBECCA	13APR58	

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

Note: You cannot update a SYSTEM 2000 database directly by using the DATA step, but you can update a SYSTEM 2000 database by using the following procedures: APPEND, FSEDIT, FSVIEW, QUEST, and SQL. For more information, see [Chapter 5, “Browsing and Updating SYSTEM 2000 Data,”](#) on page 41.

Performance Considerations

Usually, you can treat view descriptors like SAS data files in SAS programs. However, here are some things that you should consider. There are some circumstances when it is

better to extract SYSTEM 2000 data and place it in a SAS data file rather than to read it directly:

- If you plan to use the same SYSTEM 2000 data in several procedures over a period of time, you might improve performance by extracting. SAS data files are organized to provide optimal performance with PROC and DATA steps. SAS programs using SAS data files often use less CPU time than when they read SYSTEM 2000 data directly.
- If you plan to read large amounts of data from a large SYSTEM 2000 database and the database is being shared by several users (Multi-User mode), direct reading of the data could adversely affect all users' response time.
- If you are the owner of a database, and you think that reading this data directly would present a security risk, you might want to extract the data and not distribute information about either the access descriptor or the view descriptor.
- If you intend to use the data in a specific sorted order several times, it is usually best to run the SORT procedure on the view descriptor, and use the OUT= option. This is more efficient than requesting the same order of sorting repeatedly (with an ORDER BY clause) on the SYSTEM 2000 data. You cannot run PROC SORT on a view descriptor unless you use the OUT= option in the PROC SORT statement.
- Sorting data can be resource-intensive, whether it is done with PROC SORT, with a BY statement (that generates an ordering-clause), or with an ordering-clause included in the view descriptor. You should sort data only when it is needed for your program.
- If you reference a view descriptor in SAS code and the code includes a BY statement for a variable that corresponds to an item in the SYSTEM 2000 database, the interface view engine automatically generates an ordering-clause for that variable. The ordering-clause sorts the SYSTEM 2000 data before it uses the data in your SAS program. If the SYSTEM 2000 database is very large, this sorting can affect performance.

If the view descriptor already has an ordering-clause and you specify a BY statement in your SAS code, the BY statement overrides the view descriptor's ordering-clause. When you use a BY statement in SAS code with a view descriptor, it is most efficient to use a BY variable that is associated with an indexed SYSTEM 2000 item.

- When writing SAS code and referencing a view descriptor, it is more efficient to use a WHERE statement in the code than it is to use a subsetting IF statement. The interface view engine passes the WHERE statement as a SYSTEM 2000 where-clause to the view descriptor, connecting it (using the Boolean operator AND) to any where-clause included in the view descriptor. (You can further optimize the selection criteria by using connecting strings. See [“Connecting Strings to Order Conditions” on page 133](#).) Applying a WHERE clause to the SYSTEM 2000 data might reduce the number of entries processed, which often improves performance.

For more information, see [“Creating and Using View Descriptors Efficiently” on page 78](#).

Chapter 5

Browsing and Updating SYSTEM 2000 Data

Browsing and Updating SYSTEM 2000 Data Directly from SAS	41
Browsing and Updating with SAS/FSP	42
Using SAS/FSP Procedures	42
FSBROWSE Procedure	42
FSEDIT Procedure	43
FSVIEW Procedure	44
WHERE Clauses in SAS	45
Inserting and Deleting Data Records	47
Browsing and Updating with the SQL Procedure	49
Using the APPEND Procedure	52
Appending Data Described by SAS/ACCESS View Descriptors and PROC SQL Views	52
Appending Data to a SAS Data File	52
Appending Data to SAS 7 or Later Data Files	55
Appending SAS Data to a View Descriptor	56
Browsing and Updating with the QUEST Procedure	56

Browsing and Updating SYSTEM 2000 Data Directly from SAS

The SAS/ACCESS interface to SYSTEM 2000 enables you to browse and update SYSTEM 2000 data directly from a SAS session or program. This section shows you how to use SAS procedures for browsing data and updating data described by SAS/ACCESS view descriptors. The examples given here use the database Employee, and most of the examples use the view descriptor Vlib.Emppos, which you created earlier. See [Chapter 3, “SAS/ACCESS Descriptor Files,”](#) on page 19. For the definition of the other view descriptors, see [Appendix 3, “Example Programs,”](#) on page 137.

Note: Many of the examples used here involve deleting and inserting data. Before running these examples, check with your database administrator (DBA) to be sure the data in the database is correct. The data might have been changed by previous users.

Before you can browse or update SYSTEM 2000 data, you must be able to access the data by using an appropriate password and authorities. SYSTEM 2000 has various levels of passwords and authorities that enable you to display or browse data but not update

values, or you might be able to update values but not change the definition of the database. For these examples, the SYSTEM 2000 password DEMO, which is the master password for the database Employee, is stored in the view descriptors, so that you can use the SAS procedures used in these examples to update the database. For more information about SYSTEM 2000 passwords and authorities, see [Chapter 2, “SYSTEM 2000 Software,”](#) on page 5 and [Appendix 1, “Topics for Database Administrators,”](#) on page 113.

It is also important, especially for updating data, that you have some understanding of how SYSTEM 2000 logical entries map into SAS observations. See [“Mapping Data between SAS and SYSTEM 2000”](#) on page 11.

Browsing and Updating with SAS/FSP

Using SAS/FSP Procedures

If your site has SAS/FSP software in addition to SAS/ACCESS software, you can browse and update SYSTEM 2000 data that is described by a view descriptor from within a SAS program. You might use one of three SAS/FSP procedures: FSBROWSE, FSEDIT, and FSVIEW. The FSBROWSE and FSEDIT procedures display one observation at a time. The FSVIEW procedure displays multiple observations in a tabular format (similar to the PRINT procedure). PROC FSVIEW enables you to both browse and update SYSTEM 2000 data, depending on which option you specify.

When browsing SYSTEM 2000 data using the FSVIEW procedure, remember that some values are repeated for each value of the variable: LASTNAME, FORENAME, POSITION, and PAYRATE. However, the value DAVID G. REID is stored in the database only one time. For retrievals, the results are straightforward. When updating data remember that values at higher levels in the database usually do not exist as often as they seem to. If you are using PROC FSVIEW and need to make a change in one of the values. For example, to change Adkins to Atkins, type the new information over one occurrence of the value that you want to change. With a single update operation, all matching values are corrected.

If you are using PROC FSEDIT and you want to delete an observation for David Reid remember that each observation for his positions and salary data also display his last name and first name. If you delete the observation for Assistant Programmer, the deletion would not affect the LASTNAME and FORENAME values, but the POSITION and PAYRATE values would be physically removed.

Note: You cannot use the FSBROWSE, FSEDIT, or FSVIEW procedure with an access descriptor.

FSBROWSE Procedure

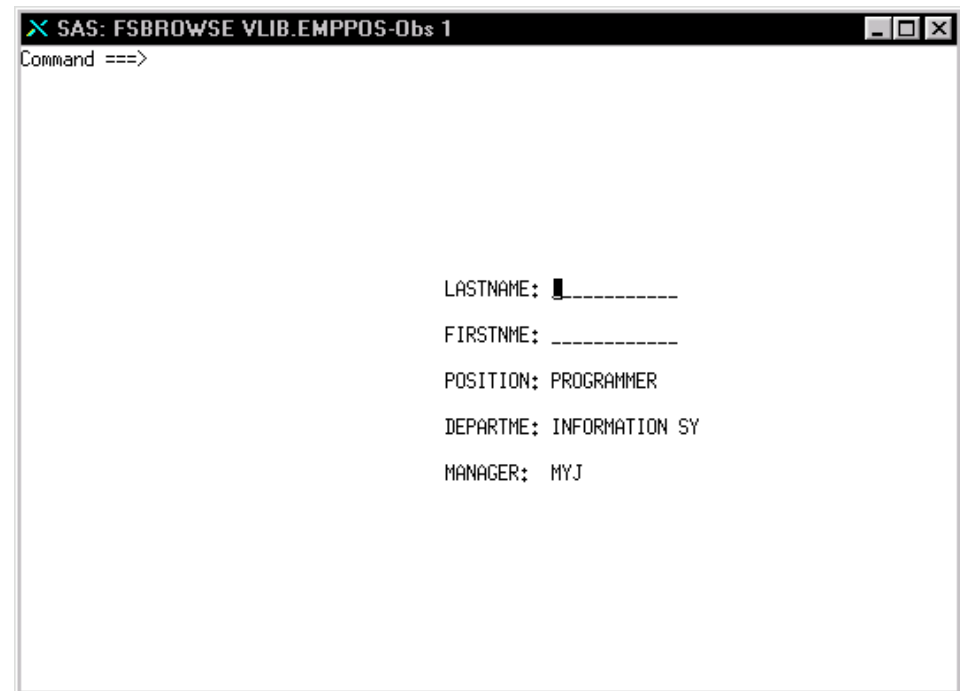
The FSBROWSE procedure enables you to look at SYSTEM 2000 data but not to change it. To use PROC FSBROWSE, submit the following:

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

PROC FSBROWSE retrieves one observation at a time from a SYSTEM 2000 database. FSBROWSE Window shows the first observation of an employee's data described by the view descriptor Vlib.Emppos. (The view descriptor contains a SYSTEM 2000 ordering-clause to order the data by last name, which is missing in the first observation. That is,

an employee has not yet been hired for that position.) To browse each observation, use the FORWARD or BACKWARD command.

Figure 5.1 FSBROWSE Window



FSEDIT Procedure

The FSEDIT procedure enables you to update SYSTEM 2000 data described by a view descriptor, if you have been granted the appropriate SYSTEM 2000 update authorities. For example, in the previous FSBROWSE window, the LASTNAME and FIRSTNME values are missing in the first observation. You can add values to these items by using PROC FSEDIT.

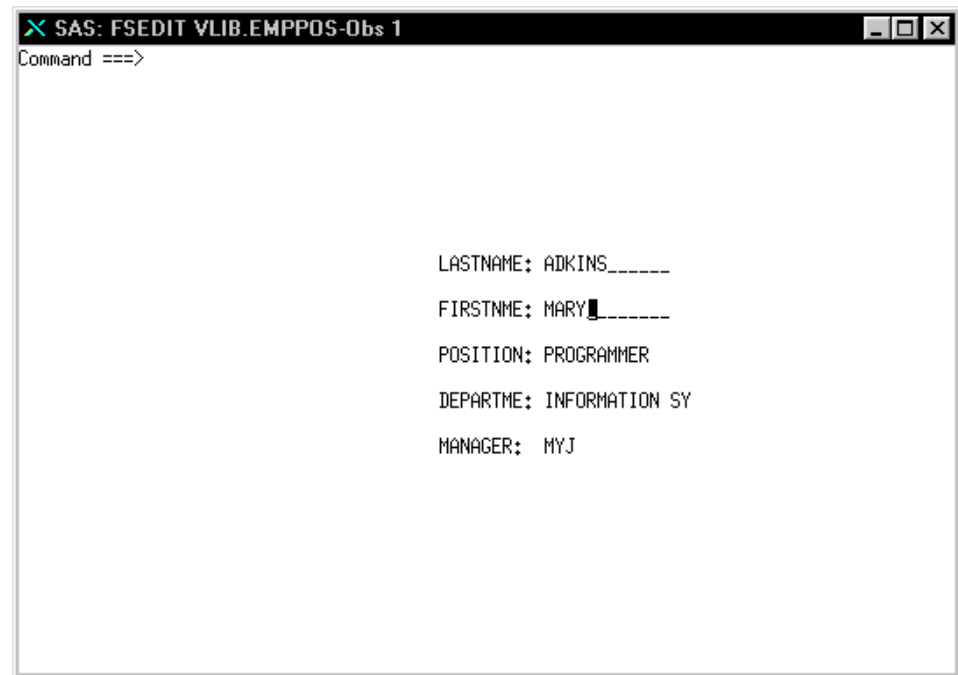
```

proc fsedit data=vlib.emppos;
run;

```

PROC FSEDIT retrieves one observation at a time. To edit data in the window, type your information. For example, for this observation, type the value 'Adkins' for LASTNAME and 'Mary' for FIRSTNME.

To end your editing session, issue the END command. To cancel an edit, you must issue the CANCEL command before you scroll to another observation. After you scroll, the change is incorporated.

Figure 5.2 FSEDIT Window

FSVIEW Procedure

The FSVIEW procedure enables you to browse or update SYSTEM 2000 data using a view descriptor, depending on how you invoke the procedure.

To browse SYSTEM 2000 data in a listing format, submit the following:

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

Browse mode is the default for PROC FSVIEW. In the FSVIEW window title in Figure 5.3, notice the (B) that follows the view descriptor's name, which indicates browse mode. Also notice that the name Mary Adkins appears, reflecting the update that you made by using PROC FSEDIT.

Figure 5.3 FSVIEW Window

Obs	LASTNAME	FIRSTNAME	POSITION	DEPARTME	MANAGER
1	ADKINS	MARY	PROGRAMMER	INFORMATION SY	MYJ
4	BOWMAN	HUGH E.	EXECUTIVE V.P.	CORPORATION	CPW
5	BROOKS	RUBEN R.	JR SALES REPRES	MARKETING	MAS
6	BROWN	VIRGINIA P.	MANAGER WESTERN	MARKETING	DMG
7	CAHILL	JACOB	MANAGER SYSTEMS	INFORMATION SY	JBM
8	CANADY	FRANK A.	MANAGER PERSONNE	ADMINISTRATION	SQT
9	CHAN	TAI	SR SALES REPRES	MARKETING	TZR
10	COLLINS	LILLIAN	MAIL CLERK	ADMINISTRATION	SQL
11	FAULKNER	CARRIE ANN	SECRETARY	CORPORATION	JBM
12	FERNANDEZ	SOPHIA	STANDARDS & PROC	INFORMATION SY	JLH

To edit SYSTEM 2000 data in a listing format, you must add the MODIFY option to the PROC FSVIEW statement, as follows:

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

The same window as shown in FSBROWSE window appears, except the window title NOW contains an (E), which indicates edit mode. For information about editing data using the FSVIEW procedure, see *SAS/FSP 9.3 Procedures Guide*.

Note: The CANCEL command does not work in the FSVIEW window.

WHERE Clauses in SAS

You can use a WHERE statement with the SAS/FSP procedure statements to specify conditions that subset the retrieved SYSTEM 2000 data. After you have invoked one of the SAS/FSP procedures, you can use a WHERE command to subset retrieved SYSTEM 2000 data.

It is more efficient to use a WHERE clause rather than a subsetting IF statement. The interface view engine translates a WHERE clause into SYSTEM 2000 conditions and passes the conditions to SYSTEM 2000 software, connecting them by default using a Boolean AND, to any SYSTEM 2000 where-clause included in the view descriptor. A where-clause in SYSTEM 2000 can reference items contained in a view descriptor and items contained in the access descriptor that the view descriptor is based on. Unlike a where-clause in SYSTEM 2000 that is stored in a view descriptor, a WHERE clause in SAS is restricted to items contained in the view descriptor.

Whether using a WHERE clause in SAS or a where-clause in SYSTEM 2000, specifying selection criteria works essentially like filters. That is, more data goes into the clauses than comes out. Using the SAS/ACCESS interface, you can pass data through more than one filter. Each filter does part of the subsetting. This is called *successive filtering*.

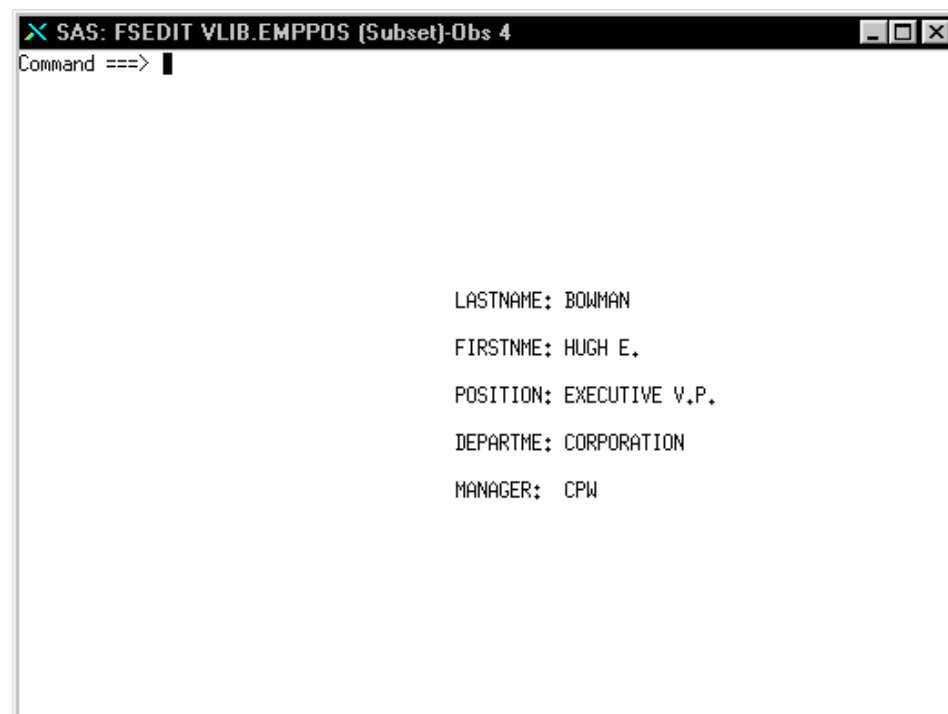
Sometimes, the interface view engine cannot translate all the WHERE clause conditions in SAS into SYSTEM 2000 conditions. In such cases, the engine subsets what it can. As partially filtered records are passed back to SAS, SAS automatically re-applies the entire WHERE clause as a second filter. This is called *post-processing*. For more information, see “WHERE Clauses in SAS” on page 45 and “where-clause in SYSTEM 2000” on page 72.

In some SAS procedures, such as PROC FSEDIT, you can continue to apply more filters by using the WHERE command on the command line in SAS. Each time you enter another WHERE clause, the process of combining and filtering conditions is repeated. The interface engine decides what conditions it can handle, connects them by default to the prior conditions by using the Boolean operator AND, sends them to SYSTEM 2000 for the first (sometimes only) filtering, and then tells SAS to do any final filtering as required. For more information, see “WHERE Clauses in SAS” on page 45 and “where-clause in SYSTEM 2000” on page 72.

In the following example, the subset of retrieved employee data comes from the Corporation Department, that is, the executives. Figure 5.4 shows the FSEDIT window after you submit the following program. Notice that the word (Subset) appears in the window title to indicate that the data that is retrieved is a subset of the data that is described by the view descriptor. Eleven observations that have the value CORPORATION for DEPARTME are retrieved for editing.

```
proc fsedit data=vlib.emppos;
  where departme='CORPORATION';
run;
```

Figure 5.4 FSEDIT Window — WHERE Clause



If you subset the data from within the procedure with the following command,

```
where departme='CORPORATION'
```

the results would be identical except that the window title would show WHERE ..., instead of (Subset), to indicate that a filter had been applied.

Although these examples have shown how to use a WHERE clause with PROC FSEDIT, you can also use WHERE clauses with PROC FSBROWSE and PROC FSVIEW. For more information about the WHERE statement in SAS, see *SAS 9.3 Statements: Reference* and *SAS Language Reference: Concepts*. For more information about the WHERE command in SAS/FSP procedures, see the *SAS/FSP 9.3 Procedures Guide*.

Inserting and Deleting Data Records

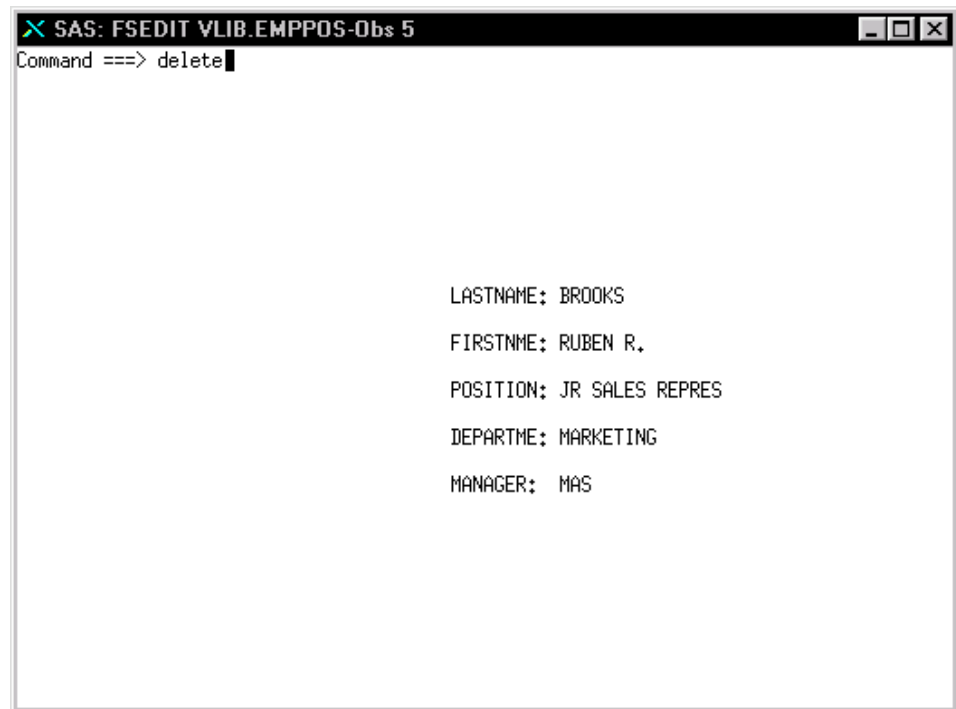
When you insert or delete data in a SYSTEM 2000 database by using a SAS/FSP procedure, be aware that the updates have the potential of affecting more than one data record in the database.

If you insert a new observation, it can cause more than one SYSTEM 2000 data record to be inserted based on how many levels the new observation represents and on a comparison between the data being inserted and the data in the last observation read, if any. During an insert, levels having data that is different from the prior observation, if any, cause a data record to be inserted. Based on how many fields you change, one or more records are inserted at the levels that have changed. If your application inserts records in a random fashion (for example, you want to add a position record for one employee while looking at the data for another employee) you should specify a BYKEY in your view descriptor. For more information about inserting data records and using a BYKEY to resolve ambiguous inserts, see [Appendix 2, “Advanced Topics for Users,” on page 121](#).

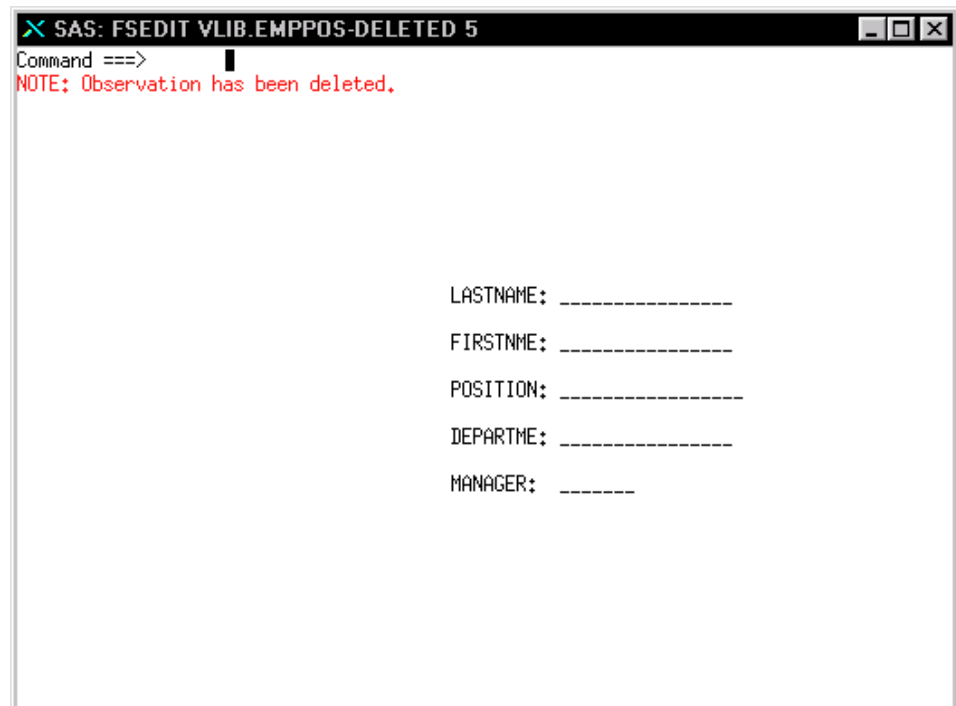
If you delete an observation, the results are not obvious to you and might be difficult to predict. The interface view engine must handle deletes carefully to ensure that the data that you request to be deleted does not adversely affect another user of the database. When you issue the DELETE command, you can expect one of the following results:

- At the very least, the items in the lowest-level record of your view descriptor are set to null (missing).
- At the very most, all the data records in the observation are physically removed from the database.
- Between those two results, the interface view engine makes a case-by-case decision on each record in the view. If the record has descendants, it is not affected. If the record has no descendants, it is physically removed.

The following example shows how to edit the SYSTEM 2000 data by deleting an observation, which is described by Vlib.Emppos. If you have been granted update authority, you can use the PROC FSEDIT statement, scroll forward to the observation that you want to delete, and issue the DELETE command from the command line, as shown in Figure 5.5.

Figure 5.5 FSEDIT Window — DELETE Command

The DELETE command processes the deletion and displays a message as shown in Figure 5.6. The observation that you deleted is no longer available for processing.

Figure 5.6 FSEDIT Window — Observation Deleted

Even though it looks as if the entire observation is removed from the database, the records are not physically removed because the POSITION WITHIN COMPANY record has descendant records that would be affected by removal. The interface view engine sets the values for the lowest-level items (POSITION, DEPARTME, and MANAGER) to missing; the values for LASTNAME and FORENAME are not affected because they are at level 0 and have descendant records. Also, values for other items in the POSITION WITHIN COMPANY record are not affected.

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

Browsing and Updating with the SQL Procedure

The SQL procedure in SAS enables you to retrieve and update data from SYSTEM 2000 databases. You must have update authority in order to edit SYSTEM 2000 data. To retrieve and browse SYSTEM 2000 data, specify a view descriptor by using a SELECT statement in PROC SQL. To update the data, you can specify view descriptors in the INSERT, DELETE, and UPDATE statements in PROC SQL. The following is a summary of these PROC SQL statements:

DELETE

deletes values from a SYSTEM 2000 database.

INSERT

inserts values in a SYSTEM 2000 database.

SELECT

retrieves and displays data from SYSTEM 2000 databases. A SELECT statement is usually referred to as a query because it queries the database for information.

UPDATE

updates values in a SYSTEM 2000 database.

The query in the following program retrieves and displays values in the SYSTEM 2000 database Employee. These values are described by the view descriptor Vlib.Emppos, if the previous updates using the SAS/FSP procedures have occurred (that is, you added the values Mary and Adkins to the programmer position.) Notice that PROC SQL prints the variable labels instead of the SAS variable names, and the data is displayed in the SAS output window. Notice also that the SELECT statement executes without using a RUN statement. PROC SQL executes when you submit it and displays output data automatically, without your having to use the PRINT procedure. The following output displays the results.

```
proc sql;
  title 'SYSTEM 2000 Data Output Using
        a SELECT Statement';
select *
/* Asterisk indicates 'select all items' */
from vlib.emppos;
```

Output 5.1 SYSTEM 2000 Data Output Using a PROC SQL Query

SYSTEM 2000 Data Output Using a SELECT Statement				
LAST NAME	FORENAME	POSITION TITLE	DEPARTMENT	MANAGER
ADKINS	MARY	PROGRAMMER	INFORMATION SY	MYJ
AMEER	DAVID	JR SALES REPRESE	MARKETING	VPB
AMEER	DAVID	SR SALES REPRESE	MARKETING	VPB
BOWMAN	HUGH E.	EXECUTIVE VICE-P	CORPORATION	CPW
BROOKS	RUBEN R.	JR SALES REPRESE	MARKETING	MAS
BROWN	VIRGINA P.	MANAGER WESTERN	MARKETING	OMG
CAHILL	JACOB	MANAGER SYSTEMS	INFORMATION SY	JBM
CANADY	FRANK A.	MANAGER PERSONNE	ADMINISTRATION	PRK
CHAN	TAI	SR SALES REPRESE	MARKETING	TZR
COLLINS	LILLIAN	MAIL CLERK	ADMINISTRATION	SQT
FAULKNER	CARRIE ANN	SECRETARY	CORPORATION	JBM

As in the SAS/FSP procedures, you can specify a WHERE clause in the SELECT statement to subset the observations that you want to display. The following program requests data about employees who are technical writers. Notice that the PROC SQL statement is not repeated in this query because you do not need to repeat the PROC SQL statement unless you use another SAS procedure or DATA step between PROC SQL statements. Because you are referencing a view descriptor, you use the SAS names for items in the WHERE clause. The following output shows the data about the one employee who is a technical writer.

```

title 'SYSTEM 2000 Data Output Subset by a
WHERE Clause';
select *
from vlib.emppos
where position='TECHNICAL WRITER';

```

Output 5.2 SYSTEM 2000 Data Output Subset by a WHERE Clause

SYSTEM 2000 Data Output Subset by a WHERE Clause				
LAST NAME	FORENAME	POSITION TITLE	DEPARTMENT	MANAGER
GIBSON	MOLLY I.	TECHNICAL WRITER	INFORMATION SY	JC

You can use the UPDATE statement to update SYSTEM 2000 data. Remember that when you reference a view descriptor in a PROC SQL statement, you are not updating the view descriptor. You are updating the SYSTEM 2000 data described by the view descriptor. Therefore, if Mary Adkins, whose name you previously added to the unfilled programmer position, decided to change her position from programmer to technical writer, you could update the information about her position title and manager by using the following program. The following output displays the results.

```

update vlib.emppos
set position='TECHNICAL WRITER'
where lastname='ADKINS';
update vlib.emppos
set manager='JC'
where lastname='ADKINS';

```

```

title 'Updated VLIB.EMPPOS View Descriptor';
select *
  from vlib.emppos;

```

Output 5.3 Updated Vlib.Emppos View Descriptor

Updated VLIB.EMPPOS View Descriptor					1
LAST NAME	FORENAME	POSITION TITLE	DEPARTMENT	MANAGER	
ADKINS	MARY	TECHNICAL WRITER	INFORMATION SY	JC	
AMEER	DAVID	SR SALES REPRESE	MARKETING	VPB	
AMEER	DAVID	JR SALES REPRESE	MARKETING	VPB	
BOWMAN	HUGH E.	EXECUTIVE VICE-P	CORPORATION	CPW	
BROOKS	RUBEN R.	JR SALES REPRESE	MARKETING	MAS	
BROWN	VIRGINA P.	MANAGER WESTERN	MARKETING	OMG	
CAHILL	JACOB	MANAGER SYSTEMS	INFORMATION SY	JBM	
CANADY	FRANK A.	MANAGER PERSONNE	ADMINISTRATION	PRK	
CHAN	TAI	SR SALES REPRESE	MARKETING	TZR	
COLLINS	LILLIAN	MAIL CLERK	ADMINISTRATION	SQT	
FAULKNER	CARRIE ANN	SECRETARY	CORPORATION	JBM	

You can use the INSERT statement to add values to a SYSTEM 2000 database or the DELETE statement to remove values as described by a view descriptor. In the following program, the values described by the view descriptor Vlib.Emppos for the employee whose last name is Adkins are deleted from the database Employee. The following output displays the results.

```

delete from vlib.emppos
  where lastname='ADKINS';

title 'Data Deleted from SYSTEM 2000 EMPLOYEE
Database';
select *
  from vlib.emppos;

```

Output 5.4 Vlib.Emppos Data with an Observation Deleted

Data Deleted from SYSTEM 2000 EMPLOYEE Database					1
LAST NAME	FORENAME	POSITION TITLE	DEPARTMENT	MANAGER	
ADKINS	MARY				
AMEER	DAVID	SR SALES REPRESE	MARKETING	VPB	
AMEER	DAVID	JR SALES REPRESE	MARKETING	VPB	
BOWMAN	HUGH E.	EXECUTIVE VICE-P	CORPORATION	CPW	
BROOKS	RUBEN R.	JR SALES REPRESE	MARKETING	MAS	
BROWN	VIRGINA P.	MANAGER WESTERN	MARKETING	OMG	
CAHILL	JACOB	MANAGER SYSTEMS	INFORMATION SY	JBM	
CANADY	FRANK A.	MANAGER PERSONNE	ADMINISTRATION	PRK	
CHAN	TAI	SR SALES REPRESE	MARKETING	TZR	
COLLINS	LILLIAN	MAIL CLERK	ADMINISTRATION	SQT	
FAULKNER	CARRIE ANN	SECRETARY	CORPORATION	JBM	

CAUTION:

You must use the WHERE clause in the DELETE statement. If you omit the WHERE clause from a DELETE statement in PROC SQL, you delete all the data in the database accessed by the view descriptor.

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

Using the APPEND Procedure

Appending Data Described by SAS/ACCESS View Descriptors and PROC SQL Views

You can use the APPEND procedure to append data that is described by SAS/ACCESS view descriptors and PROC SQL views to SAS data files. You can also update the data described by a view descriptor by appending to it the data from another SAS data set.

For an append operation to be successful, the variables in the BASE= (target) data set and the variables in the DATA= (source) data set must match, or you must use the FORCE= option to concatenate the data sets. The FORCE= option causes PROC APPEND to drop the extra variables and issues a warning..

You can append the data described by a view descriptor to a SAS 6 or later data file and vice versa. For variables that use the longer naming conventions in SAS 7 and later or for variables that otherwise do not match, use the RENAME= data set option in PROC APPEND to rename the variables.

Appending Data to a SAS Data File

In the following example, two managers have kept separate employee telephone lists. The Marketing manager kept records in the SYSTEM 2000 database Employee, which is described by the view descriptor Vlib.EmpPhon. The Corporation manager kept records for the executive telephone list in the SAS 6 SAS data file MyData.CorPhon. The two sources must be combined to create a telephone list of employees in both departments.

The data that is described by the view descriptor Vlib.EmpPhon and the data in the SAS data file MyData.CorPhon are displayed in the two outputs following the example.

```
proc print data=vlib.empphon;
    title 'Marketing Phone List';
run;

proc print data=mydata.corphon;
    title 'Corporation Phone List';
run;
```


Output 5.5 Data Described by Vlib.EmpPhon

Marketing Phone List				1
OBS	LASTNAME	FIRSTNME	PHONE	
1	AMEER	DAVID	545 XT495	
2	BROOKS	RUBEN R.	581 XT347	
3	BROWN	VIRGINA P.	218 XT258	
4	CHAN	TAI	292 XT331	
5	GARRETT	OLAN M.	212 XT208	
6	GIBSON	GEORGE J.	327 XT703	
7	GOODSON	ALAN F.	323 XT512	
8	JUAREZ	ARMANDO	506 XT987	
9	LITTLEJOHN	FANNIE	219 XT653	
10	RICHARDSON	TRAVIS Z.	243 XT325	
11	RODRIGUEZ	ROMUALDO R	243 XT874	
12	SCHOLL	MADISON A.	318 XT419	
13	SHROPSHIRE	LELAND G.	327 XT616	
14	SMITH	JERRY LEE	327 XT169	
15	VAN HOTTEN	GWENDOLYN	212 XT311	
16	WAGGONNER	MERRILEE D	244 XT914	
17	WILLIAMSON	JANICE L.	218 XT802	

Output 5.6 Data in MyData.CorPhon

Corporation Phone List				1
OBS	LASTNAME	FIRSTNME	PHONE	
1	BOWMAN	HUGH E.	109 XT901	
2	FAULKNER	CARRIE ANN	132 XT417	
3	GARRETT	OLAN M.	212 XT208	
4	KNAPP	PATRICE R.	222 XT 12	
5	KNIGHT	ALTHEA	213 XT218	
6	MILLSAP	JOEL B.	131 XT224	
7	MUELLER	PATSY	223 XT822	
8	NATHANIEL	DARRYL	118 XT544	
9	SALAZAR	YOLANDA	111 XT169	
10	WATERHOUSE	CLIFTON P.	101 XT109	

To combine the data described by these two sources, use PROC APPEND, as shown in the following program. The following output displays the data in the updated data file MyData.CorPhon. Notice that the combined data is sorted by last name. Also, because PROC PRINT was used to display the data, the variable names are used (for example, FIRSTNME), not the variable labels, which are the item names (for example, FORENAME).

```
proc append base=mydata.corphon data=vlib.empphon;
run;

proc sort data=mydata.corphon;
by lastname;

proc print data=mydata.corphon;
title 'Corporation and Marketing Phone List';
```

```
run;
```

Output 5.7 Appended Data

Corporation and Marketing Phone List				1
OBS	LASTNAME	FIRSTNME	PHONE	
1	AMEER	DAVID	545 XT495	
2	BOWMAN	HUGH E.	109 XT901	
3	BROOKS	RUBEN R.	581 XT347	
4	BROWN	VIRGINA P.	218 XT258	
5	CHAN	TAI	292 XT331	
6	FAULKNER	CARRIE ANN	132 XT417	
7	GARRETT	OLAN M.	212 XT208	
8	GARRETT	OLAN M.	212 XT208	
9	GIBSON	GEORGE J.	327 XT703	
10	GOODSON	ALAN F.	323 XT512	
11	JUAREZ	ARMANDO	506 XT987	
12	KNAPP	PATRICE R.	222 XT 12	
13	KNIGHT	ALTHEA	213 XT218	
14	LITTLEJOHN	FANNIE	219 XT653	
15	MILLSAP	JOEL B.	131 XT224	
16	MUELLER	PATSY	223 XT822	
17	NATHANIEL	DARRYL	118 XT544	
18	RICHARDSON	TRAVIS Z.	243 XT325	
19	RODRIGUEZ	ROMUALDO R	243 XT874	
20	SALAZAR	YOLANDA	111 XT169	
21	SCHOLL	MADISON A.	318 XT419	
22	SHROPSHIRE	LELAND G.	327 XT616	
23	SMITH	JERRY LEE	327 XT169	
24	VANHOTTEN	GWENDOLYN	212 XT311	
25	WAGGONNER	MERRILEE D	244 XT914	
26	WATERHOUSE	CLIFTON P.	101 XT109	
27	WILLIAMSON	JANICE L.	218 XT802	

PROC APPEND also accepts a WHERE= data set option or a WHERE statement to subset the observations from the DATA= data set that is added to the BASE= data set, as shown in the following program. (It is assumed that the data file MyData.CorPhon is in its original state before executing PROC APPEND in the preceding program.) The following output displays the results.

```
proc append base=mydata.corphon
  data=vlib.empphon(where=(lastname='AMEER'));
run;

proc print data=mydata.corphon;
  title2 'Appended Data with a WHERE= Data Set Option';
run;
```

Output 5.8 Appended Data with a WHERE= Data Set Option

Appended Data with a WHERE= Data Set Option				1
OBS	LASTNAME	FIRSTNME	PHONE	
1	BOWMAN	HUGH E.	109 XT901	
2	FAULKNER	CARRIE ANN	132 XT417	
3	GARRETT	OLAN M.	212 XT208	
4	KNAPP	PATRICE R.	222 XT 12	
5	KNIGHT	ALTHEA	213 XT218	
6	MILLSAP	JOEL B.	131 XT224	
7	MUELLER	PATSY	223 XT822	
8	NATHANIEL	DARRYL	118 XT544	
9	SALAZAR	YOLANDA	111 XT169	
10	WATERHOUSE	CLIFTON P.	101 XT109	
11	AMEER	DAVID	545 XT495	

Appending Data to SAS 7 or Later Data Files

In the previous example, if the Corporation manager kept records in a SAS 7 data file named V7.CorPhon (see the following output) and used variable names longer than eight characters, the data in Vlib.EmpPhon could be appended by using the following program:

```
proc append base=v7.corphon
  (rename (firstname=firstnme))
  data=vlib.empphon;
run;

proc sort data=v7.corphon;
  by lastname;

proc print data=v7.corphon;
  title2 'Corporation and Marketing Phone List';
run;
```

Output 5.9 Data in V7.CorPhon

Corporation Phone List			
Obs	lastname	firstname	phone
1	BOWMAN	HUGH E.	109 XT901
2	FAULKNER	CARRIE ANN	132 XT417
3	GARRETT	OLAN M.	212 XT208
4	KNAPP	PATRICE M.	222 XT 12
5	KNIGHT	ALTHEA	213 XT218
6	MILLSAP	JOEL B.	131 XT224
7	MUELLER	PATSY	223 XT822
8	NATHANIEL	DARRYL	118 XT544
9	SALAZAR	YOLANDA	111 XT169
10	WATERHOUSE	CLIFTON P.	101 XT109

In this example, the RENAME= data set option is used to reconcile a character-length discrepancy between the FIRSTNAME variable in the V7 data file and the FIRSTNME

variable in the view descriptor. The following output shows a portion of the data in the updated data file V7.CorPhon.

Output 5.10 Data in V7.CorPhon with Appended Data from Vlib.EmpPhon

Corporation and Marketing Phone List			
Obs	lastname	firstnme	phone
1	AMEER	DAVID	545 XT495
2	BOWMAN	HUGH E.	109 XT901
3	BROOKS	RUBEN R.	581 XT347
4	BROWN	VIRGINA P.	218 XT258
5	CHAN	TAI	292 XT331
6	FAULKNER	CARRIE ANN	132 XT417
7	GARRETT	OLAN M.	212 XT208
8	GARRETT	OLAN M.	212 XT208
9	GIBSON	GEORGE J.	327 XT703
10	GOODSON	ALAN F.	323 XT512
11	JUAREZ	ARMANDO	506 XT987
12	KNAPP	PATRICE R.	222 XT 12
13	KNIGHT	ALTHEA	213 XT218
14	LITTLEJOHN	FANNIE	219 XT653
15	MILLSAP	JOEL B.	131 XT224
16	MUELLER	PATSY	223 XT822
17	NATHANIEL	DARRYL	118 XT544
18	RICHARDSON	TRAVIS Z.	243 XT325
19	RODRIGUEZ	ROMUALDO R	243 XT874
20	SALAZAR	YOLANDA	111 XT169
21	SCHOLL	MADISON A.	318 XT419
22	SHROPSHIRE	LELAND G.	327 XT616
23	SMITH	JERRY LEE	327 XT169
24	VANHOTTEN	GWENDOLYN	212 XT311
25	WAGGONNER	MERRILEE D	244 XT914
26	WATERHOUSE	CLIFTON P.	101 XT109
27	WILLIAMSON	JANICE L.	218 XT802

Appending SAS Data to a View Descriptor

When appending SAS data to a view descriptor, you cannot sort the data unless you specify an output data file. To sort the data in the view descriptor, you would have to sort the SYSTEM 2000 database, which is not recommended.

For more information about the APPEND procedure, see the *SAS SQL Procedure User's Guide*.

Browsing and Updating with the QUEST Procedure

Use the QUEST procedure to access a SYSTEM 2000 database directly, that is, without using a view descriptor. This procedure is basically a messenger for SYSTEM 2000 statements: When you submit a statement in PROC QUEST, SAS scans the statement and passes it to SYSTEM 2000, which executes it.

SYSTEM 2000 includes an interactive language (also named QUEST) that is used for creating, browsing, updating, and managing SYSTEM 2000 databases. PROC QUEST

gives you full access to that language, either from interactive line-mode sessions or batch mode. In effect, when you submit the PROC QUEST statement, you start a SYSTEM 2000 session. When you submit the EXIT statement, you end the session.

Because the QUEST language is interactive, SYSTEM 2000 responds to each statement as soon as you submit it. As in PROC SQL, you do not need a RUN statement.

In this example, management is considering a reorganization and a list of all managers is requested. That information is available in the database Employee, which can be accessed in Multi-User mode. The following program uses PROC QUEST to browse and update a SYSTEM 2000 database.

```
proc quest s2kmode=m;
```

A message appears in the Log window, which verifies that you have accessed SYSTEM 2000. Now, submit SYSTEM 2000 statements to specify your password for the database and to open the database.

```
user, demo;
data base name is employee;
```

Request a list of managers by using the TALLY statement in SYSTEM 2000.

```
tally manager;
```

To end the SYSTEM 2000 session and print your report, submit the following:

```
exit;
```

The following output displays the results.

Output 5.11 TALLY Statement Output

```
*****
ITEM-      MANAGER
*****
OCCURRENCES  VALUE
-----
```

1	AFG
3	CPW
2	FAC
3	GVH
5	HEB
2	ILP
4	JBM
3	JC
1	JFS
2	JLH
1	MAS
3	MYJ
4	OMG
3	PQ
3	PRK
1	RMJ
3	SQT
4	TZR
7	VPB

```
-----
19 DISTINCT VALUES
-----
55 TOTAL OCCURRENCES
-----
```

Now, suppose that Olan Garrett, the Vice-President for Marketing, wants to make one change in his department. He decides to have Jerry Lee Smith report to a different manager. Again, use PROC QUEST to access the database Employee.

```
proc quest s2kmode=m;
user, demo; data base name is employee;
```

Request a list of all Marketing employees and their current managers by using the LIST statement in SYSTEM 2000. The following output displays the results.

```
list employee number, last name, forename, manager,
ordered by manager
where department eq marketing at 1;
```

Output 5.12 LIST Statement Output

* EMPLOYEE NUMBER	LAST NAME	FORENAME	MANAGER

* 1313	SMITH	JERRY LEE	AFG
* 1217	RODRIGUEZ	ROMUALDO R	GVH
* 1077	GIBSON	GEORGE J.	GVH
* 1133	WILLIAMSON	JANICE L.	GVH
* 1327	BROOKS	RUBEN R.	MAS
* 1011	VAN HOTTEN	GWENDOLYN	OMG
* 1161	RICHARDSON	TRAVIS Z.	OMG
* 1007	BROWN	VIRGINIA P.	OMG
* 1017	WAGGONNER	MERRILEE D	TZR
* 1119	GOODSON	ALAN F.	TZR
* 1234	SHROPSHIRE	LELAND G.	TZR
* 1031	CHAN	TAI	TZR
* 1050	AMEER	DAVID	VPB
* 1145	JUAREZ	ARMANDO	VPB
* 1015	SCHOLL	MADISON A.	VPB
* 1062	LITTLEJOHN	FANNIE	VPB

After looking at the report, Olan Garrett decides to have Jerry Lee Smith report to Madison Scholl. To do this, submit the following SYSTEM 2000 statement:

```
change manager eq mas* wh employee number eq 1313;
```

SYSTEM 2000 issues a message that one record was selected to be changed.

To end the SYSTEM 2000 session, submit the following:

```
exit;
```

Note: The commands QUIT and END are aliases for EXIT.

Chapter 6

Creating and Loading SYSTEM 2000 Databases

DBLOAD Procedure in SAS and SYSTEM 2000	59
Using the DBLOAD Procedure	59
Compatibility with SAS 6	59
Creating a SYSTEM 2000 Database	60
Loading the Input Data File	60
Subsetting Input Data	62
Subsetting Input Data	62
Loading a SYSTEM 2000 Database	63
Adding New Logical Entries versus Updating Existing Logical Entries	65
Selecting a Processing Mode for Loading Data	66

DBLOAD Procedure in SAS and SYSTEM 2000

Using the DBLOAD Procedure

The DBLOAD procedure runs in batch and interactive line mode and enables you to create and load a SYSTEM 2000 database from a SAS data set. You can create the database definition only and execute one or more incremental loads at a later time.

PROC DBLOAD constructs SYSTEM 2000 statements to create the database definition. You can load new logical entries into the database, or you can insert new records into existing logical entries. The data can come from a SAS data file, from a view created by using the SQL procedure, or from a SYSTEM 2000 database or another DBMS (using a view created by using the ACCESS procedure).

PROC DBLOAD associates each SAS variable in the input data with a SYSTEM 2000 item and assigns a default name, number, and type to each item. By default, items are non-key at level 0. You can use the defaults or change the names, the status (key or non-key), and the level, as necessary. When you are finished customizing the items, PROC DBLOAD creates the SYSTEM 2000 database.

Compatibility with SAS 6

You can use SAS 6 and later SAS data files to create and load SYSTEM 2000 databases by using PROC DBLOAD. However, beginning with SAS 7 data files, variables with names that are longer than eight characters have their names truncated to eight characters in the access and view descriptors created by PROC DBLOAD. The RENAME

statement in PROC DBLOAD can be used to rename the variables in the SYSTEM 2000 database, but it does not change the variable names in the access and view descriptors. The truncated names must be used to access the data described by the view descriptors.

Creating a SYSTEM 2000 Database

In this section, PROC DBLOAD is used to create the database Banking and load data into it. In this new database, each logical entry contains data for one customer.

The ENTRY schema record at level 0 contains two items: the customer name and customer ID. The schema record at level 1 contains information about the customer's checking and savings accounts. The schema record at level 2 contains information about the transactions for each account.

After you create the database Banking, use the QUEST procedure to execute the DESCRIBE statement in SYSTEM 2000 to produce the following output, which shows the database definition.

Output 6.1 Banking Database Definition

```

SYSTEM RELEASE NUMBER 12.1
DATA BASE NAME IS      BANKING
DEFINITION NUMBER     1
DATA BASE CYCLE NUMBER 18
  1* CUSTNAME (CHAR X(20))
  2* CUSTID (CHAR X(7))
100* RECORD_LEVEL_1 (RECORD)
  101* ACCOUNT NUMBER (INTEGER NUMBER 9999 IN 100)
  102* ACCOUNT TYPE (CHAR X IN 100)
  200* RECORD_LEVEL_2 (RECORD IN 100)
    201* TRANS TYPE (CHAR X IN 200)
    202* TRANS AMOUNT (NON-KEY MONEY $9(7).99 IN 200)
    203* TRANS DATE (DATE IN 200)

```

Notice that the records are indented at three different levels to reflect the record relationships. That is, record C200 is a descendant of record C100, which is a descendant of the level 0 record.

Loading the Input Data File

The input SAS data file that is used for the examples given here is shown in [“Data File Trans.Banking” on page 155](#). If you want to run the examples, make sure that you sort the observations before you use PROC DBLOAD. Sorting the observations groups them by accounts for each customer, which produces data in the sequence required for loading the three-level database Banking.

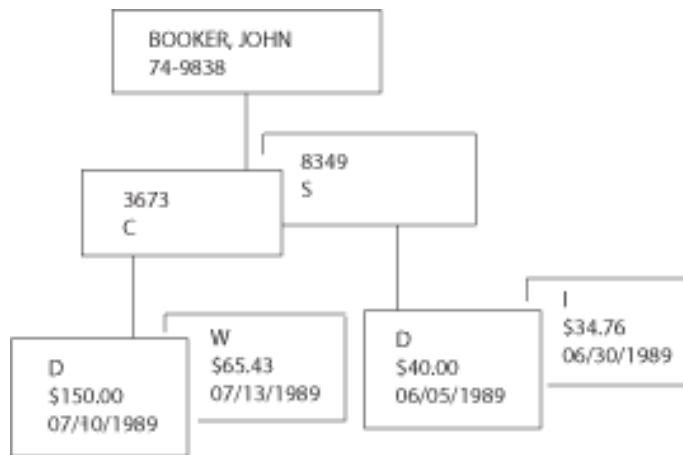
Each observation in the input data file represents one transaction. For example, John Booker has four transactions, two for each of his accounts. His name and account numbers are repeated in each observation as shown in the following output.

Output 6.2 The SAS Data File Trans.Banking

OBS	CUSTNAME	CUSTID	ACCTNUM	ACCTTYP	TRANSTYP	TRANSAMT	TRANSDAT
1	BOOKER, JOHN	74-9838	8349	S	D	\$40.00	05JUN89
2	LOPEZ, PAT	38-7274	9896	S	D	\$15.67	23JUN89
3	JONES, APRIL	85-4941	4141	C	W	\$213.78	29JUN89
4	BOOKER, JOHN	74-9838	8349	S	I	\$34.76	30JUN89
5	MILLER, NANCY	07-6163	7890	S	I	\$53.98	30JUN89
6	LOPEZ, PAT	38-7274	9896	S	I	\$16.43	30JUN89
7	JONES, APRIL	85-4941	4141	C	W	\$354.70	30JUN89
8	MILLER, NANCY	07-6163	7890	S	D	\$1,245.87	01JUL89
9	JONES, APRIL	85-4941	4141	C	D	\$2,298.65	01JUL89
10	MILLER, NANCY	07-6163	3876	C	W	\$45.98	08JUL89
11	ROGERS, MIKE	96-5052	4576	C	D	\$75.00	10JUL89
12	BOOKER, JOHN	74-9838	3673	C	D	\$150.00	10JUL89
13	LOPEZ, PAT	38-7274	9896	S	D	\$50.00	10JUL89
14	BOOKER, JOHN	74-9838	3673	C	W	\$65.43	13JUL89
15	ROGERS, MIKE	96-5052	4576	C	W	\$12.34	13JUL89
16	ROGERS, MIKE	96-5052	4576	C	W	\$45.67	13JUL89
17	MILLER, NANCY	07-6163	3876	C	D	\$56.79	14JUL89
18	ROGERS, MIKE	96-5052	4576	C	W	\$12.16	15JUL89

After you sort the input data file by customer name and account type, PROC DBLOAD loads data for each customer as a logical entry in the SYSTEM 2000 database. Redundant data is reduced, which saves storage space. The logical entry for John Booker would look like the following figure.

Figure 6.1 Sample Logical Entry in Banking Database



After you load the input data from Trans.Banking, run the following SYSTEM 2000 LIST statement in PROC QUEST.

```
list c1, c101, c102, c201, c202;
```

Here are the results.

Output 6.3 Output from LIST Statement Run on the Database Banking

* CUSTNAME	ACCOUNT NUMBER	ACCOUNT TYPE	TRANS TYPE	TRANS AMOUNT

* BOOKER, JOHN	3673	C	D	\$150.00
*			W	\$65.43
*	8349	S	D	\$40.00
*			I	\$34.76
* JONES, APRIL	4141	C	W	\$213.78
*			W	\$354.70
*			D	\$2,298.65
* LOPEZ, PAT	9896	S	D	\$15.67
*			I	\$16.43
*			D	\$50.00
* MILLER, NANCY	3876	C	W	\$45.98
*			D	\$56.79
*	7890	S	I	\$53.98
*			D	\$1,245.87
* ROGERS, MIKE	4576	C	D	\$75.00
*			W	\$12.34
*			W	\$45.67
*			W	
\$12.16				

Notice the values shown for John Booker. His name appears only one time, but he has two account numbers and four transactions. Because the examples that use PROC DBLOAD rank the data values into levels, you have a clear, logical view of the data.

Subsetting Input Data

To subset your input data, use the WHERE statement in SAS. Creating a subset of the input data is useful if you need to transfer only a portion of your SAS data to a SYSTEM 2000 database. For example, you might want to include only observations in which the value in a variable is greater than a specified number.

The following program subsets the input data to include only those observations in which the SAS variable ACCTNUM has a value greater than 4141. None of the items are renamed or indexed, and they are all at level 0.

Notice that you use the SAS variable name in the WHERE statement, not the SYSTEM 2000 item name. For information about the syntax of the WHERE statement, see *SAS Statements: Reference*.

```
proc dbload dbms=s2k data=trans.banking;
  s2kpw=mime;
  dbn=banking;
  s2kmode=m;
  where acctnum > 4141;
load;
run;
```

Subsetting Input Data

To subset your input data, use the WHERE statement in SAS. Creating a subset of the input data is useful if you need to transfer only a portion of your SAS data to a SYSTEM 2000 database. For example, you might want to include only observations in which the value in a variable is greater than a specified number.

The following program subsets the input data to include only those observations in which the SAS variable ACCTNUM has a value greater than 4141. None of the items are renamed or indexed, and they are all at level 0.

Notice that you use the SAS variable name in the WHERE statement, not the SYSTEM 2000 item name. For information about the syntax of the WHERE statement, see *SAS Statements: Reference*.

```
proc dbload dbms=s2k data=trans.banking;
  s2kpw=mime;
  dbn=banking;
  s2kmode=m;
  where acctnum > 4141;
load;
run;
```

Loading a SYSTEM 2000 Database

To create and load a SYSTEM 2000 database, use PROC DBLOAD with options and statements that describe the SYSTEM 2000 database that you want to create and the data that you want to load into the database. To load the database Banking, use the following program. The function of each statement is explained in the section that follows.

JCL statements;

```
proc dbload dbms=s2k data=trans.banking;
  s2kpw=mime;
  dbn=banking;
  accdesc=mylib.bank;
  viewdesc=vlib.myview;
  s2kmode=m;
  rename acctnum='account number' 4= 'account type'
         5='trans type' 6='trans amount'
         7='trans date';
  index 1=y 2=y 3=y 4=y transtyp=y 7=y;
  level 3=1 4=1 5=2 6=2 transdat=2;
  list all;
load;
run;
```

JCL statements;

submit your statements for execution under SAS.

proc dbload dbms=s2k data=trans.banking;

invokes the DBLOAD procedure. The DBMS= option specifies the DBMS that you want to load data into. The DATA= option specifies the SAS data file that contains the data.

s2kpw=mime;

issues the password that becomes the master password for the new database.

dbn=banking;

identifies the database that you want to create. In this example, you create a SYSTEM 2000 database named Banking.

accdesc=mylib.bank;

specifies the access descriptor libref and member name. The access descriptor is created automatically by PROC DBLOAD. In this example, the specified name is MyLib.Bank. The default access descriptor name is Work.Banking.ACCESS, where

Banking is the name of the database to be created. The access descriptor member name must not already exist when you are creating a new database.

viewdesc=vlib.myview;

specifies the view descriptor libref and member name. The view descriptor is created automatically by PROC DBLOAD. In this example, the specified name is Vlib.MyView. The default view descriptor name is Work.Banking.VIEW, where Banking is the name of the database to be created. The view descriptor member name must not already exist when you are creating a new database.

s2kmode=m;

creates the new database in a Multi-User environment. The default, S, specifies a single-user environment. For a Multi-User session, the new database files can be allocated when the session is initialized or dynamically allocated during execution by using the ALLOC command in SYSTEM 2000, Release 12.0 and later. For a single-user job, you must allocate the database files in the JCL for the job, or dynamically allocate the database files using the S2KDBCnt file.

rename acctnum='account number' 4= 'account type' 5='trans type' 6='trans amount' 7='trans date';

changes the names of the last five items. In a RENAME statement, always use a SAS variable on the left side of the equal sign (=). You can use either the SAS variable name or its positional equivalent as shown in the LIST statement output (Output 6.4). You can rename as many items as you want in one RENAME statement.

index 1=y 2=y 3=y 4=y transtyp=y 7=y;

defines items as key (indexed). In this INDEX statement, all items except TRANS AMOUNT are key items. TRANS AMOUNT is not listed, so it defaults to non-key.

In an INDEX statement, always use a SAS variable on the left side of the equal sign (=). You can use either the SAS variable name or its positional equivalent as shown in the LIST statement output below. You can index as many items as you want in one INDEX statement.

level 3=1 4=1 5=2 6=2 transdat=2;

changes the level number of an item. In this LEVEL statement, ACCOUNT NUMBER and ACCOUNT TYPE become items in a level 1 record; TRANS TYPE, TRANS AMOUNT, and TRANS DATE become items in a level 2 record. PROC DBLOAD automatically defines the schema record names RECORD_LEVEL_1 and RECORD_LEVEL_2, respectively, and assigns appropriate component numbers.

In a LEVEL statement, always use a SAS variable on the left side of the equal sign (=). You can use either the SAS variable name or its positional equivalent as shown in the LIST statement output below. You can change the level number for as many items as you want in one LEVEL statement.

list all;

lists the items, levels, and index settings.

load; run;

executes PROC DBLOAD and creates and loads the database.

Output 6.4 LIST Statement Output

```

Command ==>

PROC DBLOAD for SYSTEM 2000 - OPTIONS FOLLOW:
  Input data set=      TRANS    BANKING  DATA
  View descriptor=    VLIB    MYVIEW   VIEW
  Access descriptor=  MYLIB    BANK     ACCESS
  Database name=      BANKING
  S2KMODE=           M
  Label option=       N
  Create option=      N
  S2KLOAD=            N
-----SAS NAME---LEVEL---INDEX---COMPONENT NAME-----
  1          CUSTNAME                YES    CUSTNAME
  2          CUSTID                   YES    CUSTID
  3          ACCTNUM      1           YES    ACCOUNT NUMBER
  4          ACCTTYP      1           YES    ACCOUNT TYPE
  5          TRANSTYP     2           YES    TRANS TYPE
  6          TRANSAMT     2                    TRANS AMOUNT
  7          TRANSDAT     2           YES    TRANS DATE

```

To load additional logical entries into an existing SYSTEM 2000 database, invoke PROC DBLOAD and specify the input data file and the appropriate view descriptor. The view descriptor contains the database name, the component names, levels, and so on. It also contains the password for the database and the access mode (single-user or Multi-User). You can use a WHERE clause in SAS to limit the input. However, a SYSTEM 2000 where-clause in the view descriptor does not affect an incremental load.

To perform an incremental load with PROC DBLOAD, use the following program. In this example, the data file is Trans.IncLoad. The function of each statement is explained immediately following the program.

JCL statements;

```

proc dbload dbms=s2k data=trans.inclload;
  viewdesc=vlib.myview;
load;
run;

```

JCL statements;

submit your statements for execution under SAS.

proc dbload dbms=s2k data=trans.inclload;

invokes the DBLOAD procedure. The DBMS= option specifies the DBMS that you want to load data into. The DATA= option specifies the SAS data file that contains the data.

viewdesc=vlib.myview;

specifies the existing view descriptor libref and member name.

load; run;

executes PROC DBLOAD and loads the database.

Adding New Logical Entries versus Updating Existing Logical Entries

Incremental loads can either insert new logical entries or append new records to existing logical entries. Both types of incremental loading are performed the same way, as shown

in the previous example. How does the SYSTEM 2000 engine know which action to perform?

If you issue an S2KLOAD statement, the input observations are treated as new logical entries. Several observations can be collected to form each logical entry, but they are all new entries. The observations must be sorted in order to achieve the correct result.

If you do not issue the S2KLOAD statement, your results are based on the order of the observations and whether the view descriptor contains a BY key. A BY key identifies the placement of inserted data records in an incremental load. See [“Using a BY Key” on page 123](#). When using a BY key, it is best (less ambiguous) if your view descriptor and the BY key begin at level 0, even if you are loading records only at a lower level.

Selecting a Processing Mode for Loading Data

Two modes of processing are available when loading data with the DBLOAD procedure: insert mode and optimized load mode. Insert mode must be used to add data records to existing logical entries. Optimized load mode is a fast, efficient way to add new logical entries to the database. For details about these processing modes, see the information about the S2KLOAD statement in [“DBLOAD Procedure Statements Syntax” on page 95](#).

Chapter 7

SAS/ACCESS 9 for SYSTEM 2000: Reference

ACCESS Procedure in SAS and SYSTEM 2000	68
Types of Procedure Statements	68
Passwords for Descriptor Files	68
SAS Passwords	69
ACCESS Procedure Statements	70
ACCESS Procedure Syntax	70
The PROC ACCESS Statement	71
where-clause in SYSTEM 2000	72
Using the where-clause (SYSTEM 2000)	72
where-clause Syntax (SYSTEM 2000)	72
where-clause Examples (SYSTEM 2000)	74
ordering-clause in SYSTEM 2000	77
Using the Ordering-clause (SYSTEM 2000)	77
Ordering-clause Syntax (SYSTEM 2000)	77
ordering-clause Example (SYSTEM 2000)	77
Creating and Using View Descriptors Efficiently	78
PROC ACCESS Data Conversions	78
Dictionary	79
ASSIGN= Statement (Optional)	79
BYKEY Statement (Optional)	79
CREATE Statement (Required)	80
DATABASE Statement (Required)	81
DROP Statement (Optional)	81
FORMAT Statement (Optional)	82
INFORMAT Statement (Optional)	83
LENGTH Statement (Optional)	83
LIST Statement (Optional)	84
QUIT Statement (Optional)	85
RENAME Statement (Optional)	85
RESET Statement (Optional)	86
SELECT Statement (Optional)	87
SUBSET Statement (Optional)	88
S2KPW Statement (Optional)	89
UNIQUE Statement (Optional)	89

ACCESS Procedure in SAS and SYSTEM 2000

Types of Procedure Statements

In SAS/ACCESS, there are two categories of procedure statements: *database-description* statements and *editing* statements. The ACCESS procedure in SAS enables you to create and edit the descriptor files used by the SAS/ACCESS interface to SYSTEM 2000. Details about the ACCESS procedure statements are given in alphabetical order after the details about the PROC ACCESS statement.

Passwords for Descriptor Files

The SAS/ACCESS interface requires that access descriptors and view descriptors have a SYSTEM 2000 password to access the database. The password for an access descriptor determines the description of the database that is used to create view descriptors. The password for a view descriptor determines the data that you see, and your ability to subset and edit the data through the descriptor.

For the access descriptor, the password is specified in the DATABASE statement. For the view descriptor, the SYSTEM 2000 password is stored in the view descriptor by using the S2KPW statement, or the password can be submitted as a SAS data set option. Storing the SYSTEM 2000 password in a view descriptor, gives everyone who uses the view descriptor access to its data. Specifying a password as a data set option gives users access to the database passwords.

To protect your database passwords, store the SYSTEM 2000 password in the view descriptor, and assign one or more SAS passwords to control access to the descriptor file. You can also assign SAS passwords to control who can create view descriptors from an access descriptor. To access the descriptor files, specify the SAS password as a data set option. For example, to create a view descriptor, specify the access descriptor password in the PROC ACCESS statement after the ACCDESC= option, as follows:

```
proc access dbms=s2k accdesc=mylib.employee (alter=reward);
  create vlib.customer.view;
  select all;
run;
```

The following table summarizes the levels of protection that SAS passwords give and the effects on descriptor files.

Table 7.1 Effects of SAS Password on Descriptor Files

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

For detailed information about the levels of protection and the types of SAS passwords that you can use, see *SAS 9.3 Statements: Reference*. For more information about protecting access and view descriptors, see [“Data Security” on page 117](#).

SAS Passwords

You can assign, change, or clear a password for an access descriptor, a view descriptor, or another SAS file in SAS by using the MODIFY statement in the DATASETS procedure. The following syntax for PROC DATASETS assigns 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;
```

password-level can be one or more of the following values: READ=, WRITE=, ALTER=, or PW=. Use PW to assign Read, Write, and Alter privileges to a descriptor or a data file. *password-modification* enables you to assign a new password or to change or delete an existing password. For example, the following program assigns the password REWARD and specifies the level of protection to the access descriptor MyLib.Employe as Alter. After this program is executed, users are prompted for a password when they try to browse or edit the access descriptor or create view descriptors that are based on MyLib.Employe.

```
proc datasets library=mylib memtype=access;  
    modify employe (alter=reward);  
run;
```

In the next example, the program assigns the passwords MYPW and MYDEPT with Read and Alter levels of protection to the view descriptor Vlib.CustAcct. After this program is executed, users are prompted for the SAS password when they try to read the DBMS data, or try to browse or edit the view descriptor Vlib.CustAcct. In this instance, you need both Read and Alter levels to protect the data and the view descriptor from being read. However, a user could still update the data accessed by Vlib.CustAcct by using an UPDATE statement in PROC SQL. Assign the Write level of protection to prevent data updates.

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

To delete a password for an access descriptor or any SAS data set, put a slash after the password, as shown in the following example:

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

In the following program, PROC DATASETS sets a password for Read and Alter levels to the view descriptor Vlib.CustInfo, and PROC PRINT tries to use the view descriptor with an invalid password and, then, a valid password.

```
/* Assign passwords */  
proc datasets library=vlib memtype=view;  
    modify custinfo (read=r2d2 alter=c3po);  
run;  
  
/* Invalid password given */  
proc print data=vlib.custinfo (pw=r2dq);  
    where ssn = '178-42-6534';
```

```

    title2 'Data for 178-42-6534';
run;

/* Valid password given */
proc print data=vlib.custinfo (pw=r2d2);
    where ssn = '178-42-6534';
    title2 'Data for 178-42-6534';
run;

```

For more examples of assigning, changing, deleting, and using SAS passwords, see *SAS Statements: Reference*.

ACCESS Procedure Statements

In the SAS/ACCESS interface to SYSTEM 2000, the DATABASE statement and its options describe the database. All other statements, except CREATE, are editing statements and are optional. The DATABASE statement is specified after the CREATE statement and before any editing statements.

The options and statements that you use with PROC ACCESS depend on the task that you are performing. For example, to create an access descriptor, use the following program:

```

proc access dbms=s2k;
    create mylib.employe.access;
        DATABASE statement;
        optional editing statement(s);
run;

```

To create an access descriptor and a view descriptor, use the following program:

```

proc access dbms=s2k;
    create mylib.employe.access;
        DATABASE statement;
        optional editing statement(s);

    create vlib.emppos.view;
        optional editing statement(s);
run;

```

To create a view descriptor from an existing access descriptor, use the following program:

```

proc access dbms=s2k accdesc=mylib.employe;
    create vlib.emppos.view;
        optional editing statement(s);
run;

```

ACCESS Procedure Syntax

PROC ACCESS <options>;

```

CREATE libref.member-name.ACCESS | VIEW;
  DATABASE=database-name;
  S2KPW=password MODE= SINGLE | MULTI;
  ASSIGN= YES | NO;
  BYKEY variable-identifier = YES | NO <...variable-identifier-n = YES | NO>;
  DROP variable-identifier <...variable-identifier-n>;
  FORMAT variable-identifier = SAS-format-name
    <...variable-identifier-n = SAS-format-name>;
  INFORMAT variable-identifier = SAS-informat-name
    <...variable-identifier-n = SAS-informat-name-n>;
  LENGTH variable-identifier = item-width
    <...variable-identifier-n = item-width-n>;
  LIST ALL | VIEW | <variable-identifier>;
  QUIT;
  RENAME variable-identifier = SAS-variable-name
    <...variable-identifier-n = SAS-variable-name-n>;
  RESET ALL | variable-identifier <...variable-identifier-n>;
  SELECT ALL | variable-identifier <...variable-identifier-n>;
  SUBSET selection-criteria;
  UNIQUE = YES | NO;

```

The PROC ACCESS Statement

PROC ACCESS <*options*>;

The following options can be used in the PROC ACCESS statement:

ACCDESC= *libref.access-descriptor*

identifies an access descriptor. Use this option to create a view descriptor from an existing access descriptor.

If the access descriptor has been assigned a SAS password, you might need to specify the password in the ACCDESC= option in order to create a view descriptor based on the access descriptor. Whether you specify the password depends on the level of protection that was assigned to the access descriptor. For more information, see [“Passwords for Descriptor Files” on page 68](#).

If you create the access descriptor and the view descriptor in the same execution of PROC ACCESS, omit the ACCDESC= option because you specify the access descriptor's name in the CREATE statement.

ACCESS= and AD= are aliases.

DBMS= S2K

specifies that you want to invoke the SAS/ACCESS interface to SYSTEM 2000. This option is required when creating a descriptor, but is not required when extracting DBMS data.

OUT=*libref.member*

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 DBMS data. VIEWDESC= is used only with the OUT= option.

VIEW= and VD= are aliases.

For programs that use these options, see [Appendix 3, “Example Programs,”](#) on page 137.

where-clause in SYSTEM 2000

Using the where-clause (SYSTEM 2000)

A SYSTEM 2000 *where-clause* is used to select specific logical entries in a SYSTEM 2000 database. If the password that you are using has where-clause authority for each selected item, you might select any item included in the access descriptor from which the view descriptor is derived.

When you include a SYSTEM 2000 where-clause in a view descriptor, the selection criteria are executed each time you use the view descriptor in a SAS program. When a SYSTEM 2000 where-clause is invoked, the interface view engine:

- replaces selections of SAS variable names with database item component numbers. (The SAS variable names must correspond to a database item included in the view descriptor.)
- translates keywords to uppercase for compatibility with SYSTEM 2000.
- expands connecting strings to connect the WHERE clause in SAS to the where-clause in the view.
- preserves significant blanks in delimited text values.

The syntax of the where-clause can include one or more of the following conditions. However, you cannot include a Collect File item name or the SAME operator in a where-clause that is included in a view descriptor.

Note: This is a partial description of the SYSTEM 2000 where-clause. For a complete description, see *SYSTEM 2000 Software: QUEST Language and System-Wide Commands, Version 12, First Edition*.

where-clause Syntax (SYSTEM 2000)

WHERE *expression*;

WHERE

is the keyword that designates a where-clause. This keyword is optional if the where-clause is the first clause or if you do not specify an ordering-clause. WH is an alias.

expression might be one of the following:

- *condition*
- (*expression*)
- NOT *expression*
- *expression* AND *expression*
- *expression* OR *expression*

- *record* HAS *expression*
- *expression* AT *n*

condition [NON-KEY] *item* might be one of the following:

- unaryoperator
- binaryoperator *value*
- ternaryoperator *value* * *value*
- CONTAINS *text*
- * binaryoperator *item**

NON-KEY

enables you to change a KEY condition to a NON-KEY condition. This capability is not available in a WHERE clause in SAS. For information about using connecting strings to extend the function of the NON-KEY specification to the WHERE clause conditions in SAS, see [“Using HAS, AT, and NON-KEY” on page 135](#). NK is an alias for NON-KEY.

NOT

finds the complement of specified criteria. You can also use a logical not (\neg) symbol.

AND

combines two expressions by finding data records that satisfy both expressions. You can also use an ampersand (&).

OR

combines two expressions by finding data records that satisfy either expression or both. You can also use a vertical bar (|).

record

is a schema record name or component number.

HAS

specifies a data record by its position under its parent. This capability is not available in a WHERE clause in SAS. For information about using connecting strings to extend the function of the AT operator to the WHERE clause conditions in SAS, see [“Using HAS, AT, and NON-KEY” on page 135](#).

n

is 0 or a positive integer that indicates position of a record under its parent. The last position is indicated by 0.

item

is a schema item name or component number included in the access descriptor. You can specify a SAS variable name if the item is included in the view descriptor. The item can be KEY or NON-KEY.

unary operators: EXISTS (EXIST, EXISTING) | FAILS (FAIL, FAILING)

specifies the existence or non-existence of values.

binary operators: EQ, NE, GE, GT, LE, or LT

compares an item with a value or compares two items. You can also use the symbols shown in Table 7.2.

Table 7.2 Binary Operators and Equivalent Symbols

Operator	Equivalent Symbol
EQ	=

Operator	Equivalent Symbol
NE	\neq or $\! =$
GE	\geq or \Rightarrow or \nless or $\! <$
GT	$>$
LE	\leq or \Leftarrow or \nless or $\! >$
LT	$<$

ternary operators: EQ, NE, or SPANS (SPAN, SPANNING)

compares an item with a range of values. Ternary operators require a low value and a high value. You can also use the symbols shown in Table 7.3. There is no equivalent symbol for SPANS.

Table 7.3 Ternary Operators and Equivalent Symbols

Operator	Equivalent Symbol
EQ	$=$
NE	\neq or $\! =$

value

is a literal value or the SYSTEM 2000 system string *TODAY*. You can enclose a value with a delimiter of your choice. As shown in the following example, sometimes you might need delimiters around character values to preserve a mixed-case value. Any special character that appears at the beginning and end of a character value is assumed to be a delimiter.

```
where c1 = 'Abc De' looks for Abc De
where c1 = @Abc De@ looks for Abc De
where c1 = @Abc De looks for @Abc De
```

CONTAINS (CONT, CONTAIN, CONTAINING)

searches for characters within an item's values.

text

for the syntax and explanation of CONTAINS *text*, see *SYSTEM 2000 Software: QUEST Language and System-Wide Commands, Version 12, First Edition*.

where-clause Examples (SYSTEM 2000)

Unary Operators

Unary operators search for values that exist or do not exist using the EXISTS and FAILS operators. SYSTEM 2000 unary operators are similar to SAS missing values expressions.

The following where-clause qualifies data records that have a value for the item ACCRUED VACATION.

where accrued vacation exists

The following where-clause qualifies data records that do not have a value (that is, nulls) for the item ACCRUED VACATION.

where accrued vacation fails

Binary Operators

Binary operators compare items with a value or compare two items by using the EQ, NE, GT, GE, LT, or LE operators (or their equivalent symbols).

The following where-clause qualifies data records that have the value for EMPLOYEE NUMBER equal to 1224.

where employee number=1224

The following where-clause qualifies data records where EMPLOYEE STATUS is not equal to FULL TIME. (However, it does not qualify those records where EMPLOYEE STATUS is null.)

where employee status ne full time

The following where-clause qualifies data records where the value for HIRE DATE is greater than or equal to June 1, 1987.

where hire date=>06/01/1987

The following where-clause qualifies data records where the value for C105 equals the value for C4.

where C4 * EQ C105 *

Ternary Operators

Ternary operators search for values in a range of values by using the SPANS, EQ, and NE operators (or their equivalent symbols).

The following where-clause qualifies data records where BIRTHDAY spans the dates January 1, 1949 and January 31, 1949, inclusively.

where birthday spans 01/01/1949 * 01/31/1949

CONTAINS Operator

The CONTAINS operator searches for values that contain patterns of characters within values.

The item must be a CHARACTER, TEXT, or UNDEFINED item.

The following where-clause qualifies data records where the values for STREET ADDRESS contain the character string RIM ROCK.

where street address contains /RIM ROCK/

Combining Conditions with AND (&) and OR (|)

Using the AND and OR operators, you can combine two or more conditions. AND combines two conditions by selecting values that satisfy both conditions. OR combines two conditions by selecting values that satisfy either or both conditions.

The following where-clause qualifies data records that have COBOL in the item SKILL TYPE and 4 in the item YEARS OF EXPERIENCE.

where skill type=cobol & years of experience=4

Qualifying Unmatched Conditions with NOT (¬)

Using the NOT operator, you can select data records where values do not match a condition.

The following where-clause selects data records for the item PAY SCHEDULE that do not equal the value HOURLY or that are null.

```
where -pay schedule=hourly
```

Designating-specific Types of Records with HAS

Using the HAS operator, you can specify a focal record.

In the following where-clause, the HAS operators specify C0 (the ENTRY record) as the focal record because both conditions refer to the same schema record (C201). In this example, the HAS operators qualify C0 records that have the values COBOL and Fortran for C201. (If the HAS operator is not used, no records would qualify because there would never be a C201 value of both COBOL and Fortran.)

```
where C0 has c201 eq cobol and C0 has c201 eq fortran
```

Specifying Position in Database with AT

Using the AT operator, you can select values that are stored in a specified position in the database. Values must satisfy the condition and occupy a specific position. A data record's position is its number (reading left-to-right) below its parent record.

The following where-clause qualifies the data record in position 2 in a logical entry.

```
where position title eq programmer at 2
```

Processing Order

The order in which SYSTEM 2000 processes conditions can affect which data records are selected. SYSTEM 2000 processes conditions that have operators in the following order: AT, HAS, NOT, AND, OR.

When conditions are joined by the same operator, SYSTEM 2000 first processes KEY conditions (conditions that are indexed) from right-to-left, then NON-KEY conditions (ones not indexed) from right-to-left.

You can alter the processing order by changing the order of the conditions and by enclosing conditions in parenthesis. Conditions enclosed in parenthesis are processed first.

Because the software processes the AND operator before the OR operator, in order to access the names of employees who have an MBA degree and either a major or minor in Marketing, use the following where-clause:

```
where degree=mba &
(major field=marketing | minor field=marketing)
```

The following where-clause would also result in SYSTEM 2000 selecting the names of employees who have a minor in Marketing and degrees other than MBAs.

```
where degree=mba &
major field=marketing | minor field=marketing
```

ordering-clause in SYSTEM 2000

Using the Ordering-clause (SYSTEM 2000)

When you define a view descriptor, you can also include a SYSTEM 2000 ordering-clause to specify the order of the data. You can use only the items selected for the view descriptor. Without an ordering-clause or a BY statement in SAS, the order of the data is determined by SYSTEM 2000.

A BY statement in SAS automatically issues an ordering-clause to SYSTEM 2000. If a view descriptor already contains an ordering-clause, the BY statement overrides the ordering-clause for that program except when the SAS procedure includes the option NOTSORTED. Then, the BY statement in SAS is ignored, and the view descriptor ordering-clause is used.

Note: When you include a SYSTEM 2000 ordering-clause in a view descriptor, you can specify a terminator, either a colon (:) or a semicolon (;). If you specify both a where-clause and an ordering-clause, do not use a terminator between them.

Ordering-clause Syntax (SYSTEM 2000)

ORDERED BY *sortkeys*;

ORDERED BY

is the keyword designating an ordering-clause. ORDER BY, OB, and SORT are aliases.

sortkeys

specifies the component name, component number, or the SAS variable name of a SYSTEM 2000 item that is included in the view descriptor. Use commas to separate sort keys, which might be specified in either ascending or descending order. The default is ascending order.

- ASCENDING | ASCEND | ASC | LOW | LO specifies that you want the data ordered by ascending values of the sort key.
- DESCENDING | DESCEND | DESC | HIGH | HI specifies that you want the data ordered by descending values of the sort key.

If you specify more than one SYSTEM 2000 component, the values are ordered by the component that is named first, followed by the second component, and so on. For more information about the ordering-clause, see *SYSTEM 2000 Software: QUEST Language and System-Wide Commands, Version 12, First Edition*.

ordering-clause Example (SYSTEM 2000)

The following ordering-clause specifies that the values be sorted in ascending order based on the values in item DEPARTMENT and then, within departments, the values in item SALARY are sorted in descending order.

```
order by department, desc salary
```

Creating and Using View Descriptors Efficiently

To efficiently use SYSTEM 2000 and operating system resources:

- Select only the items your program needs. Selecting unnecessary items adds extra processing time.
- Use an ordering-clause or a BY statement in SAS to specify the order in which logical entries are presented to SAS only if SAS needs the data in a specific order for processing. (The BY statement in SAS issues an ordering-clause to SYSTEM 2000 and overrides any existing ordering-clause for the view descriptor.) If you use an ordering-clause or a BY statement in SAS, sort by an indexed item when possible.
- As an alternative to using an ordering-clause, which consumes CPU time each time you access the SYSTEM 2000 database, you can use the SORT procedure with the OUT= option to create a sorted SAS data file. This is a better approach for data that you want to use multiple times.
- If a view descriptor describes a large SYSTEM 2000 database and you use the view descriptor often, it might be more efficient to extract the data and place it in a SAS data file. (Although the extracted data file is very large, it is created only one time. However, the extracted data does not reflect any subsequent updates to the database.)
- When possible, specify selection criteria to subset the number of logical entries that SYSTEM 2000 returns to SAS.
- Write selection criteria that enable SYSTEM 2000 to use available indexes when possible. This applies whether you specify the selection criteria as part of the view descriptor or use a WHERE clause in SAS. This is especially important when accessing large databases because when SYSTEM 2000 cannot use an index, it scans the entire database sequentially.

For more information about where-clause optimization guidelines, see *SYSTEM 2000 Software: QUEST Language and System-Wide Commands, Version 12, First Edition*.

PROC ACCESS Data Conversions

Table 7.4 shows the default SAS variable formats and informats that are assigned by PROC ACCESS to each SYSTEM 2000 item type. If SYSTEM 2000 data falls outside valid SAS data ranges, an error message is printed in the SAS log when you try to read the data.

Table 7.4 Default SAS Formats and Informats for SYSTEM 2000 Item Types

SYSTEM 2000 Item Type and Picture	SAS Variable Format and Informat
CHAR X(n)	\$n
TEXT X(n)	\$CHARn.
DATE	DATE7.

SYSTEM 2000 Item Type and Picture	SAS Variable Format and Informat
INTEGER 9(<i>n</i>)	<i>n</i> .
DECIMAL 9(<i>n</i>).9(<i>d</i>)	<i>n</i> + <i>d</i> +1. <i>d</i>
MONEY 9(<i>n</i>).9(<i>d</i>)	<i>n</i> + <i>d</i> +1. <i>d</i>
REAL	BEST12.
DOUBLE	BEST12.
UNDEFINED X(<i>n</i>)	\$HEX <i>n</i> *2.

Dictionary

ASSIGN= Statement (Optional)

Generates SAS names and formats that are based on item names and data types.

Alias: AN=

Applies to: access descriptors only

Syntax

ASSIGN= YES | NO | Y | N;

Details

The ASSIGN= statement generates SAS variable names based on the first 8, non-blank characters of the item names and SAS variable attributes based on the item data types. You can change names and formats only in the access descriptor. The names saved in the access descriptor are the ones that will be used in the view descriptors.

The ASSIGN= statement causes view descriptors to inherit the SAS variable names and formats of the parent access descriptor at the time that the access descriptor is created. That is, if ASSIGN=YES (or Y), the variable names generated for the access descriptor are used in all derived view descriptors, regardless of the statements used in the view descriptor. If ASSIGN=NO (or N), you must specify the SAS variable names and formats when you create a view descriptor from this access descriptor. Use the RENAME, FORMAT, INFORMAT, LENGTH, BYKEY, and UNIQUE statements to change the variable names and attributes when creating a descriptor. The default is NO.

When a new CREATE statement is entered, the ASSIGN= statement is reset to NO.

BYKEY Statement (Optional)

Designates one or more items as sort keys.

Applies to: access descriptors and view descriptors

Syntax

BYKEY *variable-identifier* = YES | NO <...*variable-identifier-n*= YES | NO> ;

Details

The BYKEY statement designates one or more items as BY keys and, in a view descriptor, also selects them for the view.

The BYKEY statement cannot be used to change the BYKEY value in a view descriptor if ASSIGN= YES is specified in the access descriptor from which the view descriptor is derived.

variable-identifier can be one of the following:

- the current SAS name for the data item

Note: Any name on the left side of the equal sign (=) must be a SAS name, not a SYSTEM 2000 name. In an access descriptor, if the ASSIGN statement is omitted, you must enter the item number or component number (C-number) on the left side of the equal sign (=).
- a positional equivalent, which is the number that represents the item, as specified in the LIST statement
- the SYSTEM 2000 C-number of the database item

For example, if you want to make the third item a BY key, submit the following statement:

```
bykey 3=y;
```

CREATE Statement (Required)

Creates an access descriptor or a view descriptor.

Applies to: access descriptors and view descriptors

Syntax

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

Details

To create a descriptor, use a three-level name. The first level of the name is the libref of the SAS library in which you want to store the descriptor. You can store the descriptor in a temporary (Work) or permanent SAS library. The second level of the name is the access descriptor's name (that is, the member name). The third level of the name is the type of SAS file: ACCESS, for access descriptors, and VIEW, for view descriptors.

You can use the same CREATE statement to create access descriptors and view descriptors (specify the view descriptors directly following the access descriptors that they describe), unless you specify the ACCDESC= option in the PROC ACCESS statement. Then, the CREATE statement will create only view descriptors.

When you submit a CREATE statement for processing, the SAS/ACCESS interface checks the statement for errors. The descriptor is not actually written until the next CREATE or RUN statement is processed. If the SAS/ACCESS interface finds errors, error messages are written to the SAS log and processing is terminated. After you correct the error, resubmit the statements for processing.

The database-identification and DROP statements cannot be specified when creating a view descriptor.

DATABASE Statement (Required)

Specifies the SYSTEM 2000 database to use.

Alias: DB=, DBN=, or S2KDB=

Applies to: access descriptors only

Syntax

DATABASE=*database-name*;

Details

The DATABASE statement specifies the name of the SYSTEM 2000 database that you want to access. The DATABASE statement should immediately follow the CREATE statement for the access descriptor being created.

database-name can be 1 to 16 characters in length. Names longer than 16 characters are truncated and no error message appears. If the database name contains blanks or special characters, enclose the name in single or double quotation marks.

DROP Statement (Optional)

Drops the specified item so that it is not available for selection.

Applies to: access descriptors only

Syntax

DROP *variable-identifier* <...*variable-identifier-n*> ;

Details

The DROP statement drops the specified variable from the access descriptor so that the variable is not available for selection when creating a view descriptor. The specified variable in the database remains unaffected by the DROP statement.

variable-identifier can be one of the following:

- the current SAS name for the item
- the positional equivalent, which is the number that represents the item, as specified in the LIST statement
- the SYSTEM 2000 C-number of the database item.

For example, if you want to drop the third and fifth items, submit the following statement:

```
drop 3 5;
```

If you are creating an access descriptor in interactive line mode and want to mark an item as display that was previously marked as non-display with the DROP statement, use the RESET statement for that item.

Note: If you drop a record, every item in the record is dropped.

Note: If you use the RESET statement for an item, the various attributes of that item are reset (such as name, format, and so on) to their default values.

FORMAT Statement (Optional)

Assigns a SAS format to a SYSTEM 2000 data item.

Alias: FMT

Applies to: access descriptors and view descriptors

Syntax

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

Details

The FORMAT statement changes a SAS variable format from its default format; the default format is based on the database item's data type. You can enter as many formats as necessary in one FORMAT statement.

variable-identifier can be one of the following:

- the current SAS variable name for the item

Note: Any name on the left side of the equal sign (=) must be a SAS name, not a SYSTEM 2000 name. In an access descriptor, if the ASSIGN statement is omitted, you must enter the item number or component number (C-number) on the left side of the equal sign (=).
- the positional equivalent, which is the number that represents the item, as specified in the LIST statement
- the SYSTEM 2000 C-number of the database item

For example, if you want to associate the DATE9. format with the fifth item in the access descriptor, submit the following statement:

```
format 5 date9.;
```

You can use only the FORMAT statement with a view descriptor if ASSIGN= NO was specified when the access descriptor was created. When used in a view descriptor, the FORMAT statement automatically selects the reformatted item. That is, if you change the format associated with an item, you do not have to issue a SELECT statement for that item. FMT is an alias.

Note: You cannot specify the FORMAT statement for a record.

INFORMAT Statement (Optional)

Assigns a SAS informat to a SYSTEM 2000 item.

Alias: INF

Applies to: access descriptors and view descriptors

Syntax

INFORMAT *variable-identifier* = *SAS-informat-name*
 <...*variable-identifier-n*= *SAS-informat-name-n*> ;

Details

The INFORMAT statement changes a SAS variable informat from its default informat; the default informat is based on the database item's data type. You can enter as many informats as necessary using one INFORMAT statement.

variable-identifier can be one of the following:

- the current SAS variable name for the item
 - Note:* Any name on the left side of the equal sign (=) must be a SAS name, not a SYSTEM 2000 name. In an access descriptor, if the ASSIGN statement is omitted, you must enter the item number or component number (C-number) on the left side of the equal sign (=).
- the positional equivalent, which is the number that represents the item, as specified in the LIST statement
- the SYSTEM 2000 C-number of the database item

For example, if you want to associate the DATE7. informat with the second item in the access descriptor, submit the following statement:

```
informat 2 DATE7.;
```

You can use only the INFORMAT statement with a view descriptor if ASSIGN= NO in the access descriptor from which the view is derived. When used for a view descriptor, the INFORMAT statement automatically selects the reformatted item. That is, if you change the informat associated with an item, you do not have to issue a SELECT statement for that item.

Note: You cannot specify the INFORMAT statement for a record.

LENGTH Statement (Optional)

Assigns a character width to a data item.

Alias: S2KLEN or LEN

Applies to: access descriptors and view descriptors

Syntax

LENGTH *variable-identifier* = *item-width* <...*variable-identifier-n* = *item-width-n*> ;

Details

The LENGTH statement changes the item width in characters from the default width; the default item width is based on the database item's picture specification. The LENGTH statement enables SAS to handle S2K CHARACTER and TEXT items that overflow their widths (SAS does not permit variable-length character variables).

item-width can be a maximum of 200 characters.

variable-identifier can be one of the following:

- the current SAS name for the item

Note: Any name on the left side of the equal sign (=) must be a SAS name, not a SYSTEM 2000 name. In an access descriptor, if the ASSIGN statement is omitted, you must enter the item number or component number (C-number) on the left side of the equal sign (=).

- a positional equivalent, which is the number that represents the item, as specified in the LIST statement
- the SYSTEM 2000 C-number of the database item.

You can use only the LENGTH statement with a view descriptor if ASSIGN= NO in the access descriptor from which the view descriptor is derived. When used for a view descriptor, the LENGTH statement automatically selects the reformatted item. That is, if you change the length associated with an item, you do not have to issue a SELECT statement for that item.

Note: You cannot specify a LENGTH statement for a record.

LIST Statement (Optional)

Lists all or selected items in the descriptor and information about the items.

Applies to: access descriptors and view descriptors

Syntax

LIST <ALL | VIEW|*variable-identifier*> ;

Details

The LIST statement lists all or selected items in the descriptor and attributes of the items, including their positional equivalents, SYSTEM 2000 component numbers, default SAS variable names based on the first eight non-blank characters of the SYSTEM 2000 item names, and the default SAS formats based on the SYSTEM 2000 data types.

The LIST information is written to your SAS log. However, the SYSTEM 2000 item names are not listed in the log because they can be 40 or more characters in length.

You can use one or more of the following in the LIST statement:

ALL

lists all items and item attributes available for selection in the access descriptor. If an item is dropped when the access descriptor is being created, *NON-DISPLAY* is shown next to the item's description. If an item is selected when a view descriptor is being created, *SELECTED* is shown next to the item's description. If you do not specify an argument, the default is ALL.

VIEW

lists all items and item attributes in the access descriptor that is selected for the view descriptor and any subsetting or ordering criteria. VIEW is valid only when creating a view descriptor.

variable-identifier can be one of the following:

- the current SAS name for the item
- the positional equivalent, which is the number that represents the item, as specified in the LIST statement
- the SYSTEM 2000 C-number of the database item

For example, if you want to list information about the fifth item in the database, submit the following statement:

```
list 5;
```

If you want to list all of the items in the database followed by the items selected for the view descriptor, submit the following statement:

```
list all view;
```

Note: If you specify a record in a LIST statement, all the data items in that record are listed.

QUIT Statement (Optional)

Terminates the procedure without any further descriptor creation.

Applies to: access descriptors and view descriptors

Syntax

```
QUIT;
```

Details

The QUIT statement terminates the ACCESS procedure. EXIT is an alias.

RENAME Statement (Optional)

Enters or modifies the SAS name for an item.

Applies to: access descriptors and view descriptors

Syntax

```
RENAME variable-identifier = SAS-variable-name
      <...variable-identifier-n = SAS-variable-name-n> ;
```

Details

The RENAME statement enters or modifies the SAS variable name that is associated with a database item. You cannot use the RENAME= statement if ASSIGN=YES is specified in the access descriptor from which the view descriptor is derived.

When creating an access descriptor and ASSIGN=YES, you can use the RENAME statement to assign new SAS names to the default SAS names. These new names are always used when creating view descriptors based on the access descriptor.

When creating an access descriptor and ASSIGN=NO, any names assigned in the access descriptor can be changed in the view descriptor by using the RENAME statement, but the new name applies only in that view.

variable-identifier can be one of the following:

- the current SAS variable name for the item

Note: Any name on the left side of the equal sign (=) must be a SAS name, not a SYSTEM 2000 name. In an access descriptor, if the ASSIGN statement is omitted, you must use the item number or the component number (C-number) on the left side of the equal sign (=).

- the positional equivalent, which is the number that represents the item's place in the descriptor, as specified in the LIST statement.
- the SYSTEM 2000 C-number of the database item

For example, if you want to modify the SAS variable names associated with the fourth and fifth items in a descriptor, submit the following statement:

```
rename 4=hire birthday=birth;
```

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

RESET Statement (Optional)

Resets specified or all items to their default settings.

Applies to: access descriptors and view descriptors

Syntax

```
RESET ALL | variable-identifier <...variable-identifier-n> ;
```

Details

The RESET statement resets the specified items or all the items to their default values.

When creating an access descriptor, the default setting for a SAS variable name is a blank, unless you specify SAS variable names using the RENAME statement or include the ASSIGN=YES statement. When using the RESET statement, the SAS variable

names can be reset to the default name generated by PROC ACCESS (that is, the first eight characters of the variable name) or to a blank. Items dropped by using a DROP statement also become available and can be selected in view descriptors that are based on this access descriptor.

When creating a view descriptor, the results are based on the setting of the ASSIGN statement in the access descriptor from which the view descriptor is derived. If ASSIGN=YES, the RESET statement cannot be used in the view descriptor. If ASSIGN=NO and if you reset SAS variable names and variable attributes and select them later within the same procedure execution, the SAS variable names and attributes are reset to the default values generated from the item names and data types. In a view descriptor, the RESET statement clears any items specified in the SELECT statement.

You can use one or more of the following in the RESET statement:

ALL

resets all the database items defined in the access descriptor to their default names and attribute settings. When creating a view descriptor, ALL resets all the items that have been selected, so that no items are selected for the view. You can use the SELECT statement to select new items. For more information, see the SELECT statement.

variable-identifier can be one of the following:

- the current SAS name
- the positional equivalent, which is the number that represents the item, as specified in the LIST statement
- the SYSTEM 2000 C-number of the database item

For example, if you want to reset the SAS variable name and attribute associated with the third item, submit the following statement:

```
reset 3;
```

SELECT Statement (Optional)

Selects the items in the access descriptor that are to be included in the view descriptor.

Applies to: view descriptors only

Syntax

```
SELECT ALL | variable-identifier <...variable-identifier-n> ;
```

Details

The SELECT statement selects the database items in the access descriptor that you want included in the view descriptor. You might include as many items as you want in one SELECT statement.

You can use one or more of the following in the SELECT statement:

ALL

includes in the view descriptor all of the items defined in the access descriptor that were not dropped.

variable-identifier can be one of the following:

- the current SAS name
- the positional equivalent, which is the number that represents the item, as specified in the LIST statement
- the SYSTEM 2000 C-number of the database item.

For example, if you want to select the first three items, submit the following statement:

```
select 1 2 3;
```

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

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

To clear all of the current selections when creating a view descriptor, you can use the **RESET ALL** statement, and use another SELECT statement to select new items.

Note: If you select a record in a SELECT statement, all items in that record are selected.

SUBSET Statement (Optional)

Adds or modifies selection criteria defined for a view descriptor.

Applies to: view descriptors only

Syntax

SUBSET *selection-criteria*;

Details

The SUBSET statement specifies the selection criteria and ordering statement to be used by SYSTEM 2000 when creating a view descriptor. These statements are optional, but omitting them causes the view to retrieve all the data in the database. For more details about the default where-clause, see the discussion about WHERE clauses in [Appendix 2, “Advanced Topics for Users,”](#) on page 121. Here is an example of a WHERE clause:

```
subset "where amount<1010";
```

Multiple selection criteria can be included in one SUBSET statement. The quoted strings are concatenated and passed to SYSTEM 2000 for processing:

```
subset "where amount<1010"
      "ob amount";
```

To clear the selection criteria, submit the following statement:

```
subset;
```

For more information about SYSTEM 2000 where-clause and ORDER BY syntax, see the *SYSTEM 2000 Software: QUEST Language and System-Wide Commands, Version 12, First Edition* manual.

S2KPW Statement (Optional)

Stores the SYSTEM 2000 password and access mode for a view descriptor.

Applies to: view descriptors only

Syntax

S2KPW=*password* **MODE**=SINGLE | MULTI | SU | MU | S | M;

Required Argument

password

can be 1 to 4 characters in length, with no embedded blanks, and enclosed in single quotation marks. Passwords longer than 4 characters are truncated and a warning message appears. If you specify a special character for a password, it must be a single character (that is, a 1-character password) and enclosed in single quotation marks.

Optional Argument

MODE=SINGLE | MULTI | SU | MU | S | M

specifies the mode in which you want to access SYSTEM 2000. SINGLE (SU or S) means that the database in your SAS program environment is in single-user mode.

MULTI (MU or M) means that the database files are in Multi-User mode. The mode is also stored with the view. The default is MULTI.

Details

The S2KPW statement specifies the SYSTEM 2000 password and access mode for creating a view descriptor. The password that you specify is stored in encrypted form. It enables all who access the view descriptor to have access to the data that it describes. If you do not specify the S2KPW statement when creating a view descriptor, you must specify a password when using the view descriptor in order to access data from the database.

The password that is used when you open a view descriptor determines which data you see and your ability to subset and edit it through the view descriptor. You can specify the password that was used in the access descriptor from which the view is derived, or you can specify another password that encompasses a subset of the data in the view descriptor. If you specify a password that does not encompass data from the access descriptor, the view is created, but the software issues an error message when you attempt to open the view descriptor.

UNIQUE Statement (Optional)

Generates unique SAS names based on item names.

Alias: UN=

Applies to: view descriptors only

Syntax

UNIQUE= YES | NO | Y | N;

Details

The UNIQUE statement specifies whether the SAS/ACCESS interface should generate unique SAS variable names for items for which SAS variable names or variable attributes have not been specified.

You cannot use the UNIQUE statement when creating a view descriptor if ASSIGN=YES is specified in the access descriptor from which this view is derived. The YES value causes SAS to generate unique names, so the UNIQUE statement is not necessary.

If you omit the ASSIGN statement or specify ASSIGN=NO, the SAS/ACCESS interface continues to let duplicate SAS variable names exist. However, you must resolve any duplicate SAS variable names before saving (and thereby creating) the view descriptor. You can use the UNIQUE statement to automatically generate unique names, or you can use the RENAME statement to resolve duplicate names. For more information, see the [“RENAME Statement \(Optional\)”](#) on page 100.

If duplicate SAS variable names exist in the access descriptor from which this view is derived, you can specify the UNIQUE statement to resolve the duplication. Specify UNIQUE=YES to cause the SAS/ACCESS interface to append numbers to any duplicate SAS variable names, thereby making each variable name unique.

If you are running your SAS/ACCESS job in non-interactive or batch mode, it is recommended that you use the UNIQUE statement. If you do not use the UNIQUE statement and SAS encounters duplicate SAS variable names in a view descriptor, the job fails.

Chapter 8

DBLOAD Procedure Reference

DBLOAD Procedure and SYSTEM 2000	91
Creating Customized View Descriptors	92
Default SYSTEM 2000 Item Types	92
Allocating the Database Files	93
Adding Disjoint Schema Records	94
Loading One SYSTEM 2000 Database from Another	94
DBLOAD Procedure Options Syntax	94
The PROC DBLOAD Statement	94
DBLOAD Procedure Statements	94
Dictionary	95
DBLOAD Procedure Statements Syntax	95
ACCDDESC= Statement (Optional)	96
CREATE Statement (Required)	97
DBN= Statement (Required)	97
DELETE Statement (Optional)	97
INDEX Statement (Optional)	98
LABEL Statement (Optional)	98
LEVEL Statement (Optional)	99
LIST Statement (Optional)	99
LOAD Statement (Required)	100
QUIT Statement (Optional)	100
RENAME Statement (Optional)	100
RESET Statement (Optional)	101
S2KLEN Statement (Optional)	101
S2KLOAD Statement (Optional)	102
S2KMODE= Statement (Optional)	102
S2KPW= Statement (Required)	103
VIEWDESC= Statement (Required) (Optional)	103
WHERE Statement (Optional)	104

DBLOAD Procedure and SYSTEM 2000

The DBLOAD procedure enables you to create and load a SYSTEM 2000 database using data from a SAS data file, from a view created with the SQL procedure, or from a

SYSTEM 2000 database or another DBMS (using a view descriptor created by using the ACCESS procedure).

Use the DBLOAD procedure to do the following:

- create a new database definition only
- create a new database definition and load data
- load new logical entries into an existing database
- insert new data records into existing logical entries

PROC DBLOAD constructs SYSTEM 2000 statements to create a new database definition. The procedure associates each SAS variable with a SYSTEM 2000 item and assigns a default name, item type, and picture to each item. You can change the component names as necessary. Also, by default, each item is NON-KEY at level 0. However, you can change the item to be a KEY item, and you can specify a level number, which causes the procedure to create records under level 0. When you are finished customizing the items, PROC DBLOAD creates the new database definition and loads the data unless you have specified that you do not want to load any data at that time.

When you load data into an existing database, you must specify an existing view descriptor. You can specify the optimized load mode to load for new logical entries. Insert mode must be used for adding new records to existing logical entries.

PROC DBLOAD can run in batch or interactive-line mode. For efficiency, you might want to use batch mode for loads that process large amounts of data.

Creating Customized View Descriptors

When the DBLOAD procedure creates a new database, it always creates an access descriptor and a view descriptor that matches the access descriptor. The default names are *Work.database.ACCESS* and *Work.database.VIEW*.

If you do not like the default descriptors that PROC DBLOAD creates, submit a CREATE statement and invoke the ACCESS procedure to create your own specific view descriptor. Then return to PROC DBLOAD, specify the name that you created for your view descriptor, and load your data.

You must ensure that incoming SAS variables match the SYSTEM 2000 items in your view descriptor. You can have more components in your view descriptor than in the SAS data file or vice versa. PROC DBLOAD matches input variables with the variables in the view descriptor by SAS names. If your SAS names do not match, do one of the following:

- Use PROC ACCESS to create a view descriptor that matches the SAS data file.
- Use the RENAME data set option in the DATA= argument. Use the KEEP or DROP option in the DATA= argument to limit the SAS variables that are inspected.

Default SYSTEM 2000 Item Types

Table 8.1 contains the default conversions of SAS formats to SYSTEM 2000 item types. These conversions cannot be changed. However, you can alter the formats of the input

SAS variables by using the MODIFY and FORMAT statements in the DATASETS procedure if you want to affect the behavior of the type conversions. The modified formats are saved in the access and view descriptors.

If there is no SAS format, a character variable becomes the item type CHARACTER (with a picture equal to the length of the variable or the value that is specified in the S2KLEN= statement), numeric variables that are 4 bytes become the item type REAL, and numeric variables that are 8 bytes become the item type DOUBLE. The formats saved in the access and view descriptors are \$w. for character and BEST12. for numeric.

Table 8.1 Default SYSTEM 2000 Item Types and Pictures

SAS Format	SYSTEM 2000 Type
\$w.	CHAR X(n)
\$CHARw.	TEXT X(n)
any date format	DATE
w.	INTEGER 9(n)
w.d	DECIMAL 9(n-d-1).9(d)
DOLLARw.d	MONEY 9(n-d-1).9(d)
Ew.	
if $n < 8$	REAL
if $n \geq 8$	DOUBLE
\$HEXw.	UNDEFINED X(n)

Note: n is the length of the SAS variable. The value of w is ignored.

Allocating the Database Files

In the single-user environment, you must allocate the appropriate database files in your SAS session before invoking the DBLOAD procedure. For a Multi-User environment, the database files must already exist and can be allocated when the Multi-User software is initialized or, in Release 12.0 and later of SYSTEM 2000, the files can be dynamically allocated during execution by using the ALLOC command. If a database already exists, it is not released; SYSTEM 2000 returns a message and PROC DBLOAD terminates.

Adding Disjoint Schema Records

The DBLOAD procedure enables you to have records at multiple levels, but they must be on the same path. If you have disjoint schema records, you must create the database definition outside of PROC DBLOAD. Use the ACCESS procedure to create the access and view descriptors. Then, use PROC DBLOAD to load data, one path at a time, in incremental loads.

Loading One SYSTEM 2000 Database from Another

To load one SYSTEM 2000 database from another, use a view descriptor as input. However, both databases cannot be in the same execution environment if you request optimized load processing. To load one SYSTEM 2000 database from another, one database must be in the single-user environment and the other database must be in the Multi-User environment. (Optimized load mode puts the database under exclusive use, which excludes access to other databases in that environment until exclusive use is terminated.)

DBLOAD Procedure Options Syntax

The PROC DBLOAD Statement

PROC DBLOAD <options>;

The following options can be used in the PROC DBLOAD statement:

DBMS= *database-management-system*

specifies the database management system to be accessed. If you have the SAS 7 or later SAS/ACCESS interface to SYSTEM 2000 installed on your computer, the DBMS= option defaults to S2K. If you have more than one SAS 7 or later SAS/ACCESS interface installed, you must specify DBMS=S2K to access the SYSTEM 2000 data management system.

DATA= *libref.SAS-data-set*

specifies the input data set. A SAS data set can be either a SAS data file or a SAS view. If the file is permanent, you must use its two-level name, *libref.SAS-data-set*. If you do not specify a data set in the DATA= option, the default is the last SAS data set that was created.

DBLOAD Procedure Statements

The statements that you use in the DBLOAD procedure depend on whether you are creating a new database to load data into or whether you are appending data to an existing database. The following statements are required:

- **DBN=** and **S2KPW=** when creating and loading a new database

- VIEWDESC= when appending data to an existing database
- LOAD for both loading and appending
- CREATE for creating a database without loading any data

Of the remaining statements, most are used only when creating a new database; warnings are issued if you use these statements with an existing database. The following statements (listed in alphabetical order) can be used only when creating a database:

Table 8.2 *Statements to Create a Database*

ACCDISC	INDEX	RESET
CREATE	LABEL	S2KLEN
DBN	LEVEL	S2KMODE
DELETE	RENAME	S2KPW

If a view descriptor exists, PROC DBLOAD assumes that you are adding data to an existing database. Therefore, it does not accept the preceding statements, which apply only when creating a database.

Dictionary

DBLOAD Procedure Statements Syntax

Syntax

```

PROC DBLOAD <options>;
  CREATE;
  DBN= database-name;
  ACCDESC= libref.access-descriptor;
  DELETE variable-identifier <...variable-identifier-n>;
  INDEX variable-identifier = Y | N <...variable-identifier-n= Y | N>;
  LABEL;
  LEVEL variable-identifier = n <...variable-identifier-n= n>;
  LIST list-selection;
  LOAD;
  QUIT;
  RENAME variable-identifier = name <...variable-identifier-n = name-n>;
  RESET ALL | variable-identifier <...variable-identifier-n>;
  S2KLEN variable-identifier = n <...variable-identifier-n = n>;
  S2KLOAD;
  S2KMODE= M | S;
  S2KPW= password;
  VIEWDESC= libref.view-descriptor;
  WHERE SAS-where-expression;

```

ACCDESC= Statement (Optional)

Assigns a name to the access descriptor for a new database.

Alias: ACCESS= and AD=

Applies to: New databases

Syntax

```
ACCDESC= libref.access-descriptor;
```

Details

Note: When creating a new database using the ACCDESC=statement, it must follow another statement that is used only for creating a new database. The ACCDESC= statement cannot be the first statement specified in PROC DBLOAD.

The ACCDESC= statement specifies an access descriptor (member) name for the new database. If the member name already exists, the DBLOAD procedure does not create the new database.

PROC DBLOAD always creates an access descriptor file when it creates a database. By default, the new database name is `Work.database.ACCESS`, where `database` contains the first 7 characters of the new name. PROC DBLOAD also creates a view descriptor that matches the access descriptor. (See “[VIEWDESC= Statement \(Required\) | \(Optional\)](#)” on page 103.)

CREATE Statement (Required)

Creates a database definition.

Applies to: New databases

Syntax

CREATE;

Syntax Description

The CREATE statement creates a SYSTEM 2000 database definition, but does not load any data. By default, PROC DBLOAD expects you to load the data.

DBN= Statement (Required)

Specifies the database to be created

Alias: DB=

Applies to: New databases

Syntax

DBN= *database-name*;

Syntax Description

The DBN= statement is required when you are creating a new database. It specifies the name of the database to be created. A database with the same name must not already exist. *database name* must be a valid SYSTEM 2000 database name, from 1 to 16 characters in length. Database names longer than 16 characters are truncated and no error message appears. If the database name contains embedded blanks or special characters, enclose the special characters in single quotation marks. The slash (/), colon (:), and equal sign (=) are not used.

SYSTEM 2000 uses the first 7 characters of the database name as part of the dname for the database files. Any restrictions imposed by the operating environment on dnames also apply to the database name.

For single-user jobs, you must allocate your files to your SAS session. For the Multi-User environment, the database files can be allocated when the Multi-User software is initialized or, if using Release 12.0 or later of SYSTEM 2000, the files can be dynamically allocated during execution by using the ALLOC command.

DELETE Statement (Optional)

Does not load specified variables into the new database

Applies to: New databases

Syntax

DELETE *variable-identifier*<...*variable-identifier-n*> ;

Syntax Description

The DELETE statement specifies that you want to delete (drop) the specified variables from the load. By default, all SAS variables are loaded unless you specify a DELETE statement.

variable-identifier can be either the SAS variable name or the positional equivalent in the LIST output, which is the number that represents the variable's place in the data file. For example, if you want to delete the third variable, issue the following statement:

```
delete 3;
```

You can delete as many variables as you want to in one DELETE statement. If you delete more than one variable, use spaces to separate the identifiers; do not use commas.

Note: If you delete a variable from a table, this does not change the positional equivalents of the variables. For example, if you delete the second variable, the third variable is still referenced by the number 3, not 2.

INDEX Statement (Optional)

Indicates the status of items

Applies to: New databases

Syntax

INDEX *variable-identifier* = Y | N <...*variable-identifier-n*= Y | N> ;

Syntax Description

The INDEX statement indicates the KEY or NON-KEY status of items in the SYSTEM 2000 database. Y means that the item is indexed (KEY); N means the item is not indexed (NON-KEY). The default is NON-KEY (N).

variable-identifier can be either the SAS variable name or the positional equivalent in the LIST output, which is the number that represents the variable's place in the data file.

LABEL Statement (Optional)

Causes DBMS column names to default to SAS labels

Applies to: New databases

Syntax

LABEL;

Syntax Description

The LABEL statement specifies that you want the SYSTEM 2000 item names to default to the 40-character SAS variable labels. If a variable has no label, the 8-character SAS variable name is used.

LEVEL Statement (Optional)

Specifies a number for the variable level

Applies to: New databases

Syntax

LEVEL *variable-identifier* = *n* <...*variable-identifier-n=n*> ;

Syntax Description

The LEVEL statement enables you to specify a number for the level for one or more variables that will become items in the SYSTEM 2000 database. The default is level 0. If you specify any items under level 0, PROC DBLOAD automatically defines the appropriate schema records.

variable-identifier can be either the SAS variable name or the positional equivalent in the LIST output, which is the number that represents the variable's place in the data file. *n* is an integer from 0 through 9.

LIST Statement (Optional)

Lists information about the variables to be loaded

Applies to: New and existing databases

Syntax

LIST *list-selection*;

Syntax Description

The LIST statement causes a list of information to be displayed for all input variables, along with the current options, such as KEY or NON-KEY and level number. The default destination of the list is the SAS log.

list-selection can be one or more of the following:

ALL	causes all information for the load to be listed.
FIELDS ITEMS	causes all SYSTEM 2000 items for the load to be listed. ITEMS is an alias.

variable-identifier causes only one line with the information about the specified variable to be listed. The variable-identifier can be either the SAS variable name or the positional equivalent in the LIST output, which is the number that represents the variable's place in the data file. For example, if you want to list the information for the item associated with the third SAS variable, submit the following statement:

```
list 3;
```

You can use one or more of these options in the LIST statement in any order

```
list 3 fields 4;
```

This statement lists the information for the third SAS variable, followed by all the items in the data file, followed by the information for the fourth SAS variable.

LOAD Statement (Required)

Executes the load operation

Applies to: New and existing databases

Syntax

```
LOAD;
```

Syntax Description

The LOAD statement specifies that you want to execute the DBLOAD procedure.

QUIT Statement (Optional)

Terminates the DBLOAD procedure

Applies to: New and existing databases

Syntax

```
QUIT;
```

Syntax Description

The QUIT statement specifies that you want to exit the procedure without additional processing. EXIT and END are aliases.

RENAME Statement (Optional)

Renames DBMS columns

Applies to: New databases

Syntax

```
RENAME variable-identifier= name <...variable-identifier-n= name-n> ;
```

Syntax Description

The RENAME statement specifies that you want to change the names of the SYSTEM 2000 items associated with the listed SAS variables. The new component names go into the access descriptor and the view descriptor that are created for the new database.

variable-identifier can be either the SAS variable name or the positional equivalent in the LIST output, which is the number that represents the variable's place in the data file. For example, if you want to rename the item associated with the third SAS variable, submit the following statement:

```
rename 3='employee name';
```

The name must be a valid SYSTEM 2000 component name. If the item name includes embedded blanks or invalid SAS name characters, such as the pound sign (#) or hyphen (-), you must enclose the item name in single quotation marks.

n is an integer from 1 to a maximum of 9,999.

The RENAME statement enables you to include variables that were deleted. For example, if you first submit the statement DELETE 3 and then submit RENAME 3=XYZ, the third variable will be included and assigned the name XYZ and the default item type.

If you do not use the RENAME statement, all SYSTEM 2000 item names default to the corresponding SAS names or to the SAS labels if you submitted the LABEL statement. You can list multiple variables in one RENAME statement. The RENAME statement overrides the LABEL statement for the items that are renamed.

RESET Statement (Optional)

Resets column names and data types to their default values

Applies to: New databases

Syntax

```
RESET ALL | variable-identifier <...variable-identifier-n> ;
```

Syntax Description

The RESET statement resets the items that are associated with the listed SAS variables to their defaults. You can reset multiple items in one RESET statement.

ALL	resets all items to the defaults. Deleted items are restored with default values. Item names default to SAS variable names (or labels), item types are generated from the SAS variable formats, and all items are NON-KEY at level 0. ALL specifies that all previous RENAME, DELETE, INDEX, LEVEL, and S2KLEN statements are ignored.
<i>variable-identifier</i>	can be either the SAS variable name or the positional equivalent in the LIST output, which represents the variable's place in the data file.

n is an integer that defines a specific level. If you want the tenth variable, then its value is 10. There is no range for the value of *n*. For example, if you want to reset the item associated with the third SAS variable, submit the following statement:

```
reset 3;
```

S2KLEN Statement (Optional)

Changes the SAS variable length of DBMS column names

Applies to: New databases

Syntax

S2KLEN *variable-identifier* = *n* <...*variable-identifier-n* = *n*> ;

Syntax Description

The S2KLEN statement enables you to change the SYSTEM 2000 picture for a CHARACTER or TEXT type item.

variable-identifier can be either the SAS variable name or the positional equivalent in the LIST output, which represents the variable's place in the data file.

n is an integer from 1 to 250, which is used in the definition of the new database, such as CHAR X(10). If you do not specify the length of a CHARACTER or TEXT item, the SAS variable length is used.

The main reason for changing the picture is to allow overflow when the SAS length is greater than 4. A SYSTEM 2000 picture equal to or greater than 4 enables overflow of CHARACTER or TEXT type data values. For example, if the length of a SAS variable is 80 and you set the SYSTEM 2000 picture to 4, the entire value goes into overflow.

S2KLOAD Statement (Optional)

Turns on optimized load mode processing

Applies to: New and existing databases

Syntax

S2KLOAD;

Syntax Description

The S2KLOAD statement controls whether SYSTEM 2000 uses optimized load processing. You can use the optimized load mode for the initial load or for incremental loads that involve adding new logical entries. However, if you are inserting new records into existing entries, you cannot use optimized loading because the new records are inserted under existing records. The default is insert mode.

S2KMODE= Statement (Optional)

Specifies the mode for accessing SYSTEM 2000

Alias: S2KMD=

Default: S

Applies to: New databases

Syntax

S2KMODE= M | S;

Syntax Description

The S2KMODE= statement specifies the mode for accessing SYSTEM 2000.

- M specifies the SYSTEM 2000 Multi-User mode.
- S specifies the single-user mode, that is, a database in your SAS program environment. *s* is the default.

S2KMODE= is also a data set option for input views for SAS procedures. However, you cannot use it as a data set option in the DBLOAD procedure. For more information, see [“Overriding Options” on page 121](#).

S2KPW= Statement (Required)

Assigns a database password

Applies to: New databases

Syntax

S2KPW= *password*;

Syntax Description

The S2KPW= statement specifies the master password for the database that is being created. The password must be acceptable to SYSTEM 2000.

password can be 1 to 4 characters in length, have no embedded blanks, and can be enclosed in single quotation marks. Passwords longer than four characters are truncated, and a warning message is issued. If you specify a special character for the password, it must be a single character (that is, a 1-character password) enclosed in single quotation marks.

S2KPW= is also a data set option for input views for SAS procedures. However, you cannot use it as a data set option in the DBLOAD procedure. For more information, see [“Overriding Options” on page 121](#).

VIEWDESC= Statement (Required) | (Optional)

Assigns a name to the view descriptor for a new database

Applies to: Optional for new databases , Required for existing databases

Note: When creating a new database using the VIEWDESC= statement, it must follow some statement that is only for creating a new database. The VIEWDESC= statement cannot be the first statement specified in PROC DBLOAD.

Syntax

VIEWDESC= *libref.view-descriptor*;

Syntax Description

The VIEWDESC= statement identifies the view descriptor for the SYSTEM 2000 database that is being created or loaded.

For an existing database, the VIEWDESC= statement is required because it contains the database name and identifies the password and the components in the view.

When you create a new database, PROC DBLOAD creates a view descriptor. By default, the new database name is Work.*database*.VIEW, where *database* contains the first 7 characters of the new name. The view descriptor matches the access descriptor from which it is derived. Use the VIEWDESC= statement to specify the libref and member name for a permanent view descriptor. If the member name for the view descriptor already exists, PROC DBLOAD does not create a new database.

WHERE Statement (Optional)

Subsets input data

Applies to: New and existing databases

Syntax

WHERE *SAS-where-expression*;

Syntax Description

The WHERE statement specifies how you want to subset your input data.

SAS-where-expression must be a valid WHERE statement in SAS.

Note: You must use SAS variable names in the WHERE statement; do not use the SYSTEM 2000 component names.

The following statement loads only those observations that contain JONES and APRIL in the SAS variable CUSTNAME:

```
where custname='JONES, APRIL';
```

For information about the WHERE statement in SAS, see *SAS Statements: Reference*.

Chapter 9

QUEST Procedure Reference

QUEST Procedure in SAS with SYSTEM 2000	105
Statements in PROC QUEST	106
SYSTEM 2000 Statements and the QUEST Procedure	106
Using the QUEST Procedure	106
ECHO ON and ECHO OFF Statements	107
SYSTEM 2000 Strings and Functions	107
Single-User Mode	108
Using Single-User Mode	108
The S2KPARMS File	108
Attention Interrupts in TSO	108
Multi-User Mode	109
Using Multi-User Mode	109
Temporary Output File	109
Command File	109
Attention Interrupts in TSO	110
Dictionary	110
QUEST Procedure Syntax	110
MCS Procedure Statement (Optional)	111
QUIT Procedure Statement (Optional)	111
SCS Procedure Statement (Optional)	112

QUEST Procedure in SAS with SYSTEM 2000

The QUEST procedure enables you to perform the following tasks in SYSTEM 2000 databases from within SAS:

- define new databases
- assign passwords
- retrieve data
- update data
- enable rollback
- restore a database
- save a database

PROC QUEST is interactive; SYSTEM 2000 executes statements as soon as you submit them. You do not need a RUN statement.

Note: If you issue a RUN statement, SAS ignores it when you use PROC QUEST.

SAS statements that can be issued anywhere (for example, TITLE and FILENAME) are also available when you use the QUEST procedure.

Statements in PROC QUEST

Note: All the statements in PROC QUEST are optional.

PROC QUEST statements specify how SYSTEM 2000 statements are submitted within the QUEST procedure, that is, single command submission (SCS) or multiple command submission (MCS). You can also submit SYSTEM 2000 statements in a Command File and not use any PROC SYSTEM 2000 Statements and the QUEST Procedure and Single-User Mode. QUEST statements. Multiple command submission and Command Files are supported only in the Multi-User access mode.

For more information, see “SYSTEM 2000 Statements and the QUEST Procedure” on page 106 and “Single-User Mode” on page 108.

SYSTEM 2000 Statements and the QUEST Procedure

Using the QUEST Procedure

SYSTEM 2000 statements in PROC QUEST might be any valid SYSTEM 2000 statement that is available in the Self-Contained Facility, which includes the following:

- CONTROL statements to save and restore databases, to assign passwords and authorities, to create and remove indexes, and so on,
- DEFINE statements to define, change, and delete database components in the database definition
- QUEST statements to access a database for retrieval and updates
- REPORT statements to produce customized reports

For more information, see the *Quick Reference Guide*.

Note: When you submit SYSTEM 2000 statements in PROC QUEST, the statements are subject to SAS syntax rules. For example, you must end statements with a semicolon (;) instead of a colon (:), use '/' and '/' to delimit comments, and so on. SYSTEM 2000 statements that contain a character literal that is more than 200 characters in length are rejected. If you want to use a single quotation mark (') or double (closing) quotation marks (") as the delimiter in a SYSTEM 2000 where-clause, you must use one of the techniques described below. Otherwise, quotation marks cause ambiguity between the SAS parser and the SYSTEM 2000 parser.

A quoted string is required if the SYSTEM 2000 where-clause condition contains any of the following:

- a value that is specified in mixed case

- a where-clause keyword, such as OR, AT, or AND, in the value. For example, `PRINT ENTRY WHERE C303 CONTAINS 'INSTRUCTOR AT ACC';`
- a single quotation mark (') or double (closing) quotation marks (") in the value

You might want to use quoted strings because you are accustomed to using them in other systems.

To use a single quotation mark (') or double (closing) quotation marks (") around a value in a where-clause condition, you can use either of the following methods:

- Create a short SYSTEM 2000 Command File that contains the following two statements, which make the where-clause delimiter a single quotation mark ('):

```
DELIMITER IS ' ;
COMMAND FILE IS INPUT;
```

Invoke this short Command File one time to change the delimiter; the second statement returns you to your usual way of entering statements in PROC QUEST. By running this short Command File at the beginning of a PROC QUEST session, you do not have to put every statement that contains a quoted string into a separate Command File.

Note: SYSTEM 2000 commands in a Command File must be specified in uppercase or a syntax error occurs.

- If you do not want to create the short Command File, you can submit the following statements from the Program Editor:

```
delimiter is ' ; ' ;
'x' ;
```

You will receive messages from SYSTEM 2000 and a syntax error warning, but, after the 'X' statement finishes processing, you can safely use single quotation marks (') as the where-clause delimiter.

If a value contains a single quotation mark, change the delimiter to double quotation marks (") by using one of the preceding methods.

ECHO ON and ECHO OFF Statements

SYSTEM 2000 messages are displayed in the Log window, along with SAS messages.

The ECHO ON statement specifies that echoes of SYSTEM 2000 statements appear in the Output window in addition to the statement output. This is convenient for debugging or interpreting results.

The ECHO OFF statement specifies that echoes of SYSTEM 2000 statements do not appear in the Output window.

Note: If you use ECHO ON and the MCS statement, you get one echo of all the commands at the same time no matter how many individual commands were submitted in one MCS. Also, only 249 characters are echoed.

SYSTEM 2000 Strings and Functions

You probably need to modify existing SYSTEM 2000 strings and functions when you use PROC QUEST because

- the statement terminator must be the semicolon (;), not the colon(:). Any colons embedded in the strings or functions do not work correctly. Therefore, a string or function with embedded statement terminators can be invoked from within or from outside PROC QUEST, but not both.

- statements cannot start with the default SYSTEM 2000 separator, the asterisk (*). In SAS, an asterisk signifies the beginning of a comment.
- statements cannot contain a percent sign (%) as a system separator. In SAS, a percent sign signifies the beginning of a macro statement.

To avoid problems with system separators, submit a SEPARATOR IS statement to change the separator. For example, the following statement changes a separator to a comma (,):

```
separator is ,;
```

Single-User Mode

Using Single-User Mode

When you invoke PROC QUEST in single-user mode (S2KMODE=S), SAS displays the following information:

- SYSTEM 2000 initialization parameters
- SYSTEM 2000 version number
- copyright information

Submit the USER statement to establish your password and start your SYSTEM 2000 session. The following example statements also attach the database Employee to your session:

```
user,demo;
data base name is employee;
```

The SYSTEM 2000 interface to SAS accesses the database and displays any output in the Output window or in the procedure output file (if you are executing in interactive line mode).

The S2KPARMS File

Because you are running in your own address space when you are in single-user access mode, you can specify a variety of parameters. (See *SYSTEM 2000(R) Software: Product Support Manual, Version 1, First Edition* for information about SYSTEM 2000 execution parameters.) To specify execution parameters for the QUEST procedure in single-user mode, you must set up a file with the fileref S2KParms. If there is no fileref named S2KParms, system defaults are used.

For example, the LIST=YES parameter displays the parameter values in the Log window when the system is initialized. To suppress this display, edit the file and specify LIST=NO and allocate it using the fileref of S2KParms in your SYSTEM 2000 CLIST or EXEC.

Attention Interrupts in TSO

If you interrupt where-clause processing under TSO in single-user mode, your request is canceled. A canceled retrieval produces no output, and a canceled update does not alter the database.

If you interrupt processing at any other time, the interrupt is ignored. When you return to SAS, the usual SAS interrupt-handling mechanism is restored.

While it is not recommended, you can prevent single-user mode from intercepting the SAS attention-interrupt mechanism by specifying STAX=NO in the file S2KParms. However, be aware that, if you terminate both the QUEST procedure and SAS by using the SAS Attention exit after having updated the database, you might have unwritten buffers left in memory. As a result, the database could be damaged.

Multi-User Mode

Using Multi-User Mode

When you invoke PROC QUEST in Multi-User mode (S2KMODE=M), SAS displays the following messages:

```
QUEST Ready
S2K3212/00 - SYSTEM 2000 INTERACTIVE INTERFACE READY -
```

Submit the USER statement to establish your password and start your SYSTEM 2000 session. For example, the following statements attach the database Employee to your session:

```
user,demo;
data base name is employee;
```

The SYSTEM 2000 interface to SAS accesses the database and displays any output in the Output window or in the procedure output file (if you are executing in interactive line mode).

Temporary Output File

PROC QUEST uses a disk file to temporarily store output. In CMS, the file is allocated automatically. In TSO, you need to allocate a file with at least 10 tracks of 3350 disk space (or the equivalent) and assign the ddname S2KOUTP. If the file is not allocated, the warning message -895- appears, and any response from SYSTEM 2000 that exceeds 4096 bytes is truncated.

Usually, 10 tracks of 3350 disk space is enough for typical use. If some output cannot be stored in S2KOUTP, message -897- appears, and you lose some output. Re-allocate the file with more space later.

Command File

You can save SYSTEM 2000 statements in a file by using the SAS Text Editor and directing PROC QUEST to read statements from that file by submitting the following statement:

```
local command file is fileref;
```

fileref is the ddname for the file. However, any SAS macros in the file are not expanded because PROC QUEST submits the statements directly to SYSTEM 2000 without SAS reading them.

The following statement lets you continue to submit statements from the Program Editor; write it at the end of the Command File:

```
LOCAL COMMAND FILE IS INPUT;
```

If you omit this command, PROC QUEST automatically returns to the Program Editor when it finds an end-of-file in the Command File.

Note: SYSTEM 2000 commands in a Command File must be specified in uppercase or a syntax error occurs.

PROC QUEST enables you to use alternate user files for the Data File, Message File, and Report File. They can be local files or files allocated in the Multi-User region.

Attention Interrupts in TSO

If you interrupt processing while running PROC QUEST under TSO, the usual SAS message (asking whether you want to terminate or continue) does not appear. Instead, an attention interrupt in the Output window is interpreted to mean that you want to purge any additional output. The first line on the next page is displayed, but additional output from your last statement is discarded. If you interrupt processing at any other time, the interrupt is ignored.

To cancel the session, you must first terminate PROC QUEST.

Dictionary

QUEST Procedure Syntax

Syntax

```
PROC QUEST <S2KMODE= M | S> <BLANKS>;
  MCS;
  QUIT;
  SCS;
  SYSTEM 2000 statements;
```

Optional Arguments

S2KMODE=M | S

specifies the mode in which you want to access SYSTEM 2000, M for Multi-User mode or S for single-user mode. Use M to access Multi-User SYSTEM 2000 running under a different CMS computer or z/OS address space. Use S to load and execute your own copy of SYSTEM 2000 on your computer or address space. The default is M.

S2KMD= is an alias for S2KMODE=. (ACCESS=, ACC=, DBACCESS=, and DBACC=, which were developed for SAS 5, might also be used.)

BLANKS

retains all blanks in SYSTEM 2000 statements and passes them to SYSTEM 2000 software. You can specify the BLANKS option in order to retain blanks in LIST column headings, TEXT values, and report titles and headings. If you do not specify the BLANKS option, SAS deletes extraneous blanks by default. That is, leading,

trailing, and embedded blanks are stripped from the statements before SYSTEM 2000 reads the statements. BLANK is an alias for BLANKS.

MCS Procedure Statement (Optional)

Puts the QUEST procedure into MCS mode.

Syntax

MCS;

Details

The MCS statement puts the QUEST procedure into statement-queuing mode (multiple command submission). That is, PROC QUEST accumulates statements in a 32760-byte buffer before submitting them to SYSTEM 2000. If the accumulated statements fill the buffer, the system displays message -898-. This message instructs you to submit the statements in the buffer by entering two semicolons (;;) or to submit the SCS statement to erase the buffer contents and terminate the MCS mode. You remain in MCS mode until you issue the SCS statement.

The MCS statement is ignored in single-user access mode. In a Multi-User environment, you can use the MCS statement or a Command File to submit a long sequence of SYSTEM 2000 statements that must be processed as a group. For example:

- to define a new database or modify an existing definition
- to submit statements to the Report Writer feature
- to submit a set of IF-THEN/ELSE statements
- to submit QUEUE and TERMINATE statement blocks

Issue the RECALL command to bring submitted statements back into the Program Editor. Issue the SAVE command to store the statements in an external file. Then, when you submit the LOCAL COMMAND IS statement, SYSTEM 2000 processes the statements in that file.

If you exit SAS, the MCS buffer is emptied without sending the statements to SYSTEM 2000. To exit SAS and close the database, type BYE on the command line in the Program Editor or submit an ENDSAS statement.

If you submit long strings of statements that terminate with the ENDSAS statement, you must end the SYSTEM 2000 session by using two semicolons (;;) in order to submit the statements to SYSTEM 2000.

QUIT Procedure Statement (Optional)

Terminates the QUEST procedure.

Syntax

QUIT;

Details

The QUIT statement closes the SYSTEM 2000 database, terminates the SYSTEM 2000 session, and ends the QUEST procedure. END and EXIT are aliases for QUIT. You can issue a SYSTEM 2000 EXIT statement any time during a PROC QUEST session.

SCS Procedure Statement (Optional)

Puts the QUEST procedure in SCS mode.

Syntax

```
SCS;
```

Details

The SCS statement puts PROC QUEST into single-statement queuing mode (single command submission).

Note: SCS erases the 32760-byte buffer (used with the MCS mode) even if the buffer contains statements that were not yet sent to SYSTEM 2000.

Appendix 1

Topics for Database Administrators

SYSTEM 2000 and the SAS/ACCESS Interface	113
Overview for the Database Administrator	113
SYSTEM 2000 Interface View Engine	114
Changing a SYSTEM 2000 Database Password	116
Changing a Database Definition	116
Data Security	117
Ensuring Data Security	117
SYSTEM 2000 Security	118
SAS System Security	118
Enabling the Rollback Log	118
Locking Record Levels	119
Maximizing SYSTEM 2000 Performance	120

SYSTEM 2000 and the SAS/ACCESS Interface

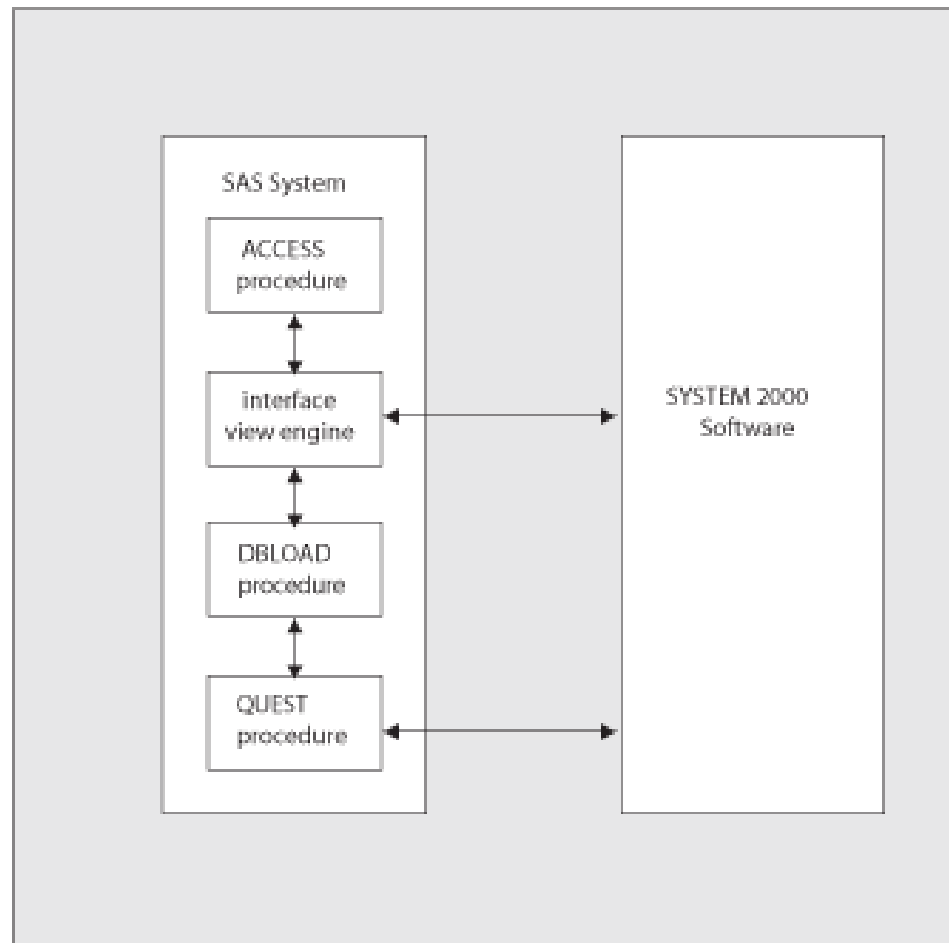
Overview for the Database Administrator

Understanding how the SAS/ACCESS interface to SYSTEM 2000 works can help a Database Administrator (DBA) decide how to use it.

When an access descriptor file is created by using the ACCESS procedure, SAS calls SYSTEM 2000 to get a description of the database. When a view descriptor file is created, SAS already has the information about the database in the access descriptor, so it does not call SYSTEM 2000.

PROC ACCESS writes the descriptor files to a SAS library. Then, when a SAS procedure is used with a view descriptor whose data is in a SYSTEM 2000 database, the SAS Supervisor calls the interface view engine to access the data. The engine can access a database for reading, updating, inserting, and deleting.

The connections between the SAS procedures and SYSTEM 2000 are shown in the following figure:

Figure A1.1 SAS Procedures and SYSTEM 2000

SYSTEM 2000 Interface View Engine

Using the SYSTEM 2000 Interface View Engine

The SYSTEM 2000 interface view engine is a PLEX (Programming Language Extension) applications program that retrieves and updates data in a SYSTEM 2000 database. Use the following to make calls to the engine:

- the ACCESS procedure to create an access descriptor
- the DBLOAD procedure and specify a view descriptor by using the VIEWDESC= option
- the QUEST procedure to negotiate an execution environment
- a SAS DATA step or SAS procedures and specify a view descriptor by using the DATA= option

In all instances, the same PLEX commands initiate and terminate communication between the interface view engine and SYSTEM 2000. Each time a different SAS procedure requires use of SYSTEM 2000, the procedure makes an initialization call to the engine. This first call establishes communication with SYSTEM 2000. Then, the interface view engine issues:

1. the START S2K command for a single-user or Multi-User environment, as specified by the calling SAS procedure.
2. an OPEN command for the specified database and then returns control to the procedure. Additional calls to the engine perform retrieval and update operations specified by the SAS procedure until the procedure is terminated.
3. a CLOSE command for the database that was opened.
4. the STOP S2K command when the entire SAS session terminates, or when you run the QUEST procedure in the same environment (single-user or Multi-User) that the SAS programs have been running in.

Using the ACCESS Procedure

The ACCESS procedure calls the interface view engine to retrieve item and record information for a specified database. The engine sends the component number, name, type, picture, level number, and key status (with the database cycle number, and so on) back to the procedure for each item and record in the database. PROC ACCESS stores this information in the access descriptor for later use when creating view descriptors. PROC ACCESS also calls the engine to extract information into a SAS data file.

Using the DBLOAD Procedure

When you create a new database, the DBLOAD procedure always creates an access descriptor and a view descriptor.

To insert data into an existing SYSTEM 2000 database, you must specify an appropriate view descriptor by using the VIEWDESC= option in the DBLOAD procedure. The view descriptor provides a mapping between the SAS variables that contain data to be inserted and the SYSTEM 2000 components that insert the data into the database. It also contains the database name, password, and access mode to be used when you insert data.

For each observation that is retrieved from the data file specified in the DATA= option, a corresponding call is made to the interface view engine. The engine inserts the data into the database identified by the view descriptor. The engine uses only Insert mode (one at a time) for inserting new descendant records into existing logical entries. Also, if the DATA= option identifies a SYSTEM 2000 view descriptor, the interface view engine is called to read that view.

When you load new logical entries (starting with records at level 0) into a database, you can specify that you want to use an optimized loading process. SYSTEM 2000 processes the new logical entries as one batch of inserts (PLEX load mode). The optimized load mode is faster than inserting records one at a time. However, it causes SYSTEM 2000 to attach the database for exclusive use, and no other database can be open in the same execution environment. Therefore, if your job is using optimized load, your input cannot also be a SYSTEM 2000 view descriptor of a database in the same environment.

Using the QUEST Procedure

Usually, the QUEST procedure communicates directly with SYSTEM 2000. When you enter SYSTEM 2000 statements (commands), they are processed immediately, and the results are sent back to you, interactively. However, there is one exception. Each time you execute the QUEST procedure, it calls the engine to request permission to execute. If no other SAS programs are using the engine in the same environment, permission is granted. Otherwise, permission to execute is refused. Similarly, when the QUEST procedure terminates, it calls the engine to signal the event so that the engine can enable other SAS programs to execute. The engine does not enable SAS jobs to execute in the same environment while the QUEST procedure is running.

Using Other SAS Procedures

SAS procedures can access records in a SYSTEM 2000 database by specifying a view descriptor in the DATA= option. SAS examines the view descriptor to determine which database management system is specified and passes control to the appropriate engine. The interface view engine uses information stored in the view descriptor (for example, access mode, password, database name, component numbers, levels, types, and so on) to process SYSTEM 2000 data records as if they were observations in a SAS data file.

Before performing retrievals, the engine processes a PLEX dynamic where-clause (if specified) to select a subset of data records that should be processed as observations. The engine constructs the dynamic where-clause from the view where-clause and the WHERE clause in SAS (if any). If no view where-clause exists, a where-clause is constructed to locate all database records.

The dynamic where-clause processing returns a Locate File that contains the addresses of database records that satisfied the selection criteria. Based on those addresses, the engine issues a combination of GET S2KCOUNT, GET1, and GETA commands to read one or more database records. Then, the engine combines data from the records (according to the view descriptor) to form a SAS observation that it passes back to the calling procedure for processing.

Based on the capabilities of the SAS procedure that you are using, the next call to the engine might be a request to update or delete the SAS observation that was just retrieved. For updates, the engine issues MODIFY, INSERT, and REMOVE commands for one or more data records, based on how many records were used to construct the observation. Then, the SAS procedure calls the engine again to retrieve another SAS observation. The engine locates another group of records, constructs another SAS observation, and returns it to the SAS procedure. This cycle continues until the SAS procedure is terminated or until the last qualified SAS observation has been constructed and returned to the SAS procedure. The interface view engine also uses other commands, such as COMMIT, ROLLBACK, and CLEAR, to control processing.

Changing a SYSTEM 2000 Database Password

The master password holder (usually, the DBA) can change any database password at any time. If a password that is stored in a view descriptor is changed, the DBA can either change the view descriptor or override the stored password each time the view descriptor is used. The software does not require that you use the same password. The engine only requires that you use a password that has enough authority to service the view descriptor.

Because SYSTEM 2000 passwords are not stored in access descriptors, there are no effects if a SYSTEM 2000 password is changed. Passwords can be stored in view descriptors, but changing a password does not affect the view descriptor. It still has all the items, but you might not be able to use the view. Nothing is automatically changed in the descriptors when you change a password or its authorities.

Changing a Database Definition

Changes to a database definition can affect view descriptor files. The interface view engine validates the view against the current definition and issues an error message if it detects discrepancies.

The following sections contain details about the effects of changes to a SYSTEM 2000 database definition on existing view descriptors.

- Changes that do not affect existing view descriptors:
 - creating or deleting indexes
 - inserting new schema items
 - deleting schema items not referenced in any view descriptor

Note: If an access descriptor includes the deleted schema item, users might create a view descriptor using that item, which would cause a problem.
 - inserting or deleting schema records in paths not referenced in any view descriptor

Note: If an access descriptor references the changed path, users might create a view descriptor using that path. This would cause a problem.
- Changes that might affect existing view descriptors:
 - changing an item name. If the item name was used in a where-clause or an ordering-clause that is stored in the view descriptor, a syntax error message appears when you try to use the view descriptor. The message indicates an unrecognized component name.
 - changing the attributes of items that are not in the view descriptor but are referenced in the stored where-clause.
- The following changes can cause existing view descriptors to fail:
 - inserting or deleting a level in the path of a view descriptor.
 - changing the attributes of an item or the component number of an item, so that it points to something different. Specifically,
 - You can change the pictures for CHARACTER, TEXT, and UNDEFINED item types, but you cannot change them to a DATE or NUMERIC item type.
 - You cannot change a DATE item type to any other item type.
 - You cannot change a NUMERIC item type to a non-numeric item type or change its picture.
 - changing the component number of parent records for any schema item or record in the path of a view descriptor.
 - deleting items that are referenced in a view descriptor.

Data Security

Ensuring Data Security

SAS preserves the data security provided by SYSTEM 2000 and SAS. The DBA controls who has SYSTEM 2000 authorities and who can create SYSTEM 2000 databases. Creators of the databases control who can access the data. Therefore, SAS users can access only SYSTEM 2000 databases that they created or databases for which they have specific password authorities.

To protect data from accidental update or deletion, you can use precautionary measures on both sides of the interface.

SYSTEM 2000 Security

In SYSTEM 2000, the DBA gives users secondary passwords that enable only the authority that they must have. For example, Jane needs to create a view descriptor that reads and selects only the personal information about each employee in the database Employee, which is stored in the ENTRY record. To do this, Jane only needs to perform retrievals and where-clause selection on schema items C1 through C16. Use the following commands to assign her authorities:

```
valid password is jane;
assign r,w to c1 through c16 for jane;
```

John needs to add new employees' names to the database, so he needs all authorities. Use the following commands to assign his authorities:

```
valid password is john;
assign r, u, w to all components for john;
```

With retrieval, update, and where-clause authorities, John can create a view descriptor that reads the data records for the schema items and can use that view descriptor to add new logical entries to the database.

If SYSTEM 2000 detects a security violation while a SAS procedure or DATA step is running, it issues the return code 45 or 47 and an error message. If rollback is enabled for the database, partial updates are rolled out (canceled).

SAS System Security

In SAS, the DBA can do the following:

- Set up all access descriptors and drop items that contain sensitive data.
- Set up all view descriptors and enable users access to them on a selective basis by storing the appropriate passwords in the descriptors, or requiring you to supply a password.
- Give users Read-Only access to the SAS library in which the access descriptors are stored. Read-Only access enables users to see only the items selected for each view descriptor and prevents them from editing access descriptors.
- Set up several access descriptors for multiple secondary passwords, or require you to create the access descriptors.

Enabling the Rollback Log

A single SAS observation can be composed of one or more SYSTEM 2000 database records. Therefore, a single UPDATE command in SYSTEM 2000 to update a given observation might involve several internal SYSTEM 2000 UPDATE commands. If one of these UPDATE commands fails after several others have executed, the status of the entire update is incomplete.

In order to guarantee the data integrity, you must enable the rollback feature. You can do this easily with the QUEST procedure, by issuing the ENABLE ROLLBACK statement in the CONTROL language. When you enable rollback, you must make sure that the

Rollback Log (database File 8) and the Update Log (database File 7) are allocated. With rollback enabled, SYSTEM 2000 can roll back the database to its status before the sequence of commands that triggered the error.

If rollback is not enabled, partial updates can occur if error return codes are received. Errors can occur from security violations or from bad data. For example, data that does not match the SAS informat, or data that has too many significant decimal places for a specific item's numerical precision. Also, if LHOLD=YES is specified in the SYSTEM 2000 execution parameters and rollback is not enabled, the interface view engine can receive return code 111, which causes an update to be rejected.

Locking Record Levels

SAS supports several levels of locking through the CNTLLEV= data set option. If CNTLLEV=REC (the default), SYSTEM 2000 performs record-level locking. The interface view engine interprets any value for the CNTLLEV= option other than REC, to signify that it should enable exclusive use of the database. Also, the database is under exclusive use if you issue the S2KLOAD statement in the DBLOAD procedure. Exclusive use locks out all other users until the database is closed, which usually occurs when the procedure ends. (The database CLOSE operation depends on the procedure used.)

Note: Some SAS procedures, such as statistical procedures, set CNTLLEV=MEM internally because multiple passes of the data must be made. For example, finding the median requires more than one pass.

In a Multi-User environment, exclusive use of a database can cause contention in a database. Also, if you have specified optimized load mode (S2KLOAD) in PROC DBLOAD, your input to that load cannot be a SYSTEM 2000 view descriptor for a database in the same environment.

When exclusive use of the database is not requested, the interface view engine uses SYSTEM 2000 record-level locking and multiple local holds. This means that an observation is locked for retrieval, and unlocked only when some other observation is retrieved or when the file is closed. Updates do not unlock an observation. Record-level locking can cause contention in a SYSTEM 2000 database. The interface view engine takes the following steps to keep the contention to a minimum:

- At retrieval time, the engine attempts to lock all records in the path (using the PLEX /HOLD option). If the lowest-level record in the path (that is, the record farthest from level 0) cannot be locked, an error return code is sent to SAS that indicates that this observation cannot be locked. Records above the lowest level in the view are locked if possible. However, the engine does not regard it as an error when they cannot be locked, and no message is sent to warn you.
- At update time, only those records that were successfully locked can be updated. For updates at levels that were not previously locked, the engine tries again to obtain the locks. If it cannot get them, the update fails, a return code indicates that it could not get the necessary lock, and partial updates are rolled back if rollback is enabled. If the engine gets the locks, it checks to verify that the data in the database is the same as when the data was originally retrieved. If the data is the same, the update takes place.

The purpose of this locking mechanism is to avoid contention. You can always access a path if the lowest-level record can be locked. You do not have to wait for another user to drop a lock onto one of the upper-level records. (However, there might be relatively few locks of upper-level records.) You are guaranteed to be able to update only items in the

lowest-level record of the view descriptor. The engine attempted to update records at any level that you specify and performed the update if it can.

Maximizing SYSTEM 2000 Performance

Among the factors that affect SYSTEM 2000 performance are the size of the database being accessed, the number of items being accessed, and the number of data records qualified by the selection criteria. For databases that have many items and many data records, you should evaluate all SAS programs that need to access the database directly. In your evaluation, consider the following:

- Does the program need all the SYSTEM 2000 items? If not, create and use an appropriate view descriptor that includes only the items that are 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 it is, consider extracting the data and placing it in a SAS data file for SAS procedures to use, instead of the data being accessed directly by each procedure. See [“Performance Considerations” on page 38](#) for circumstances when extracting data is the more efficient method.
- Do the records need to be in a specific order? If they do, include a SYSTEM 2000 ordering-clause in the appropriate view descriptors or an ORDER BY clause in a SAS program.
- Do the selection criteria enable SYSTEM 2000 to use key (indexed) items and non-key (not indexed) items efficiently? See [“where-clause in SYSTEM 2000” on page 72](#) for guidelines for specifying efficient selection criteria.
- What type of locking mechanism does SYSTEM 2000 need to use? See [“Locking Record Levels” on page 119](#).

Appendix 2

Advanced Topics for Users

Overriding Options	121
Override Corresponding Values	121
Using Multiple View Descriptors	122
Deleting Data Records	122
Inserting Data Records	123
BY Key	123
Using a BY Key	123
Examples Using a BY Key	124
BY-Key Effects on Performance	125
Missing Values (Nulls)	126
Retrieving Nulls	126
Updating Nulls	126
Nulls in Selection Criteria	127
WHERE Clauses in SAS and where-clauses in SYSTEM 2000	127
Overview of WHERE Clauses	127
WHERE Clauses in SAS Translatable to SYSTEM 2000	130
WHERE Clauses in SAS Not Translatable to SYSTEM 2000	131
NOT Operator in SAS and SYSTEM 2000	132
Specifying Selection Criteria	133
Connecting Strings to Order Conditions	133
Using Connecting Strings	133
Syntax for Specifying a Connecting String	134
Omitting a WHERE Clause in SAS	135
Using the OR Operator	135
Using HAS, AT, and NON-KEY	135
Stored Strings in SYSTEM 2000	136

Overriding Options

Override Corresponding Values

Data set options enable you to override corresponding values stored in a view descriptor. The S2KPW= and S2KMODE= data set options can be specified by using the DATA=

argument in any PROC statement except PROC DBLOAD. The options are in effect only for a single execution of the procedure.

S2KPW=*password*

enables you to override the SYSTEM 2000 password stored in the view descriptor. If no password is stored in the view descriptor, the S2KPW= option must be used to provide a valid password for the database.

The password must be an alphanumeric value that is 1 to 4 characters in length with no embedded blanks and can be enclosed in single quotation marks. Passwords longer than 4 characters are truncated and a warning message is issued. If the password is a special character, it must be a single character (that is, a 1-character password) enclosed in single quotation marks.

Use the S2KPW= option in the DATA= argument, where DATA= specifies a SYSTEM 2000 view descriptor that is used as input to a SAS procedure except PROC DBLOAD.

Note: Passwords specified in PROC DBLOAD cannot be overridden.

S2KMODE=S | M

enables you to override the SYSTEM 2000 access mode that is stored in the view descriptor. S2KMODE=S executes the procedure as a single-user job, which means that you allocate the database files in your job and execute a separate copy of SYSTEM 2000. S2KMODE=M indicates that the database files are allocated in a region controlled by the Multi-User software. S2KMD is an alias.

Use the S2KMODE= option in the DATA= argument, where DATA= specifies a SYSTEM 2000 view descriptor that will be used as input to a SAS procedure except PROC DBLOAD, which uses the mode that is specified for a new database, or the mode that is stored in the view descriptor for an incremental load.

The following program executes the FSEDIT procedure using the view descriptor EmpPos. The data set options specified in the PROC FSEDIT statement use the password DEMO and execute SYSTEM 2000 in single-user mode.

```
proc fsedit data=vlib.emppos
  (s2kmode=s s2kpw=demo);
run;
```

Using Multiple View Descriptors

You can use multiple view descriptors in a single SAS session, but only one view descriptor can be open for updating. This restriction applies to either one window that opens two view descriptors or two windows that each open one view descriptor. You cannot have the QUEST procedure and a SAS procedure or a DATA step that refers to a SYSTEM 2000 view descriptor active at the same time in two windows, unless one is single-user mode and the other is Multi-User mode.

Deleting Data Records

If you are deleting an observation from the S2K database (for example, by using the FSEDIT procedure), use the DELETE command. However, the SAS/ACCESS interface sets all the values of items in the view descriptor (that is, only the selected items in the

same record) to missing and removes the lowest-level data record from the database if one or more of its items were selected for display. Ancestor records are also removed if they do not have other descendant records. Any data records that are removed must be locked, but they are not removed until you move to a different observation.

The DELETE command does not remove items or records unless your password has U-authority for the specific items and records.

Inserting Data Records

You can insert data records with SYSTEM 2000 by using the insert or the optimized load mode when updating records with various SAS procedures or when loading a database using the DBLOAD procedure. In PROC DBLOAD, you specify the mode by using the S2KLOAD statement.

When a new observation is inserted, it can cause the insertion of more than one SYSTEM 2000 database record. The number of inserts is based on how many levels are in the database, and on a comparison between the data being inserted and the data in the last observation (if any) that was read. During an Insert operation, record levels that have data that is different from the prior observation (if any) result in a SYSTEM 2000 database record being inserted.

You must use Insert mode if you are loading new records into existing logical entries. The insert or the optimized load mode can be used in PROC DBLOAD when you are loading new logical entries. The optimized load mode is more efficient than insert mode, but the optimized load mode has the following restrictions:

- Data must be sorted in data-tree order before the load.
- Logical entries are always inserted in their entirety.
- The number of inserts and the levels at which inserts are performed are based on the order of the data and on which fields change from observation to observation.
- Your input cannot be a SYSTEM 2000 view in the same environment.

Note: During optimized load processing, your output database is open in exclusive use mode with rollback temporarily disabled.

Insert mode is suitable for mass insertion of descendant records into existing logical entries when using PROC DBLOAD. Similar to the optimized load mode, the interface view engine determines where to insert the new records, based on the values of fields in the observation. When you insert an observation, the engine compares it to the prior observation. Based on how many fields have changed, one or more records are inserted at the levels that have changed. Also, you can use a BY key to help determine where records are inserted. (BY keys are discussed in the next section.)

BY Key

Using a BY Key

A *BY key* is similar to a BY group in SAS, which groups observations based on one or more fields. Many SAS procedures process records in BY groups. Also, some updates in the DATA step are performed by matching specified BY variables in different data sets.

A similar matching process occurs with BY-key items in the SAS/ACCESS interface to SYSTEM 2000. Use the BY-key capability to eliminate redundancy and to help the interface view engine find an existing path for inserting the new records.

Each time the interface view engine is called to insert an observation, it inspects the changes that you made from observation to observation, in order to determine how many data records to insert into the database.

If none of the data changed, or if the changes were only at the lowest level of the view, the engine needs to insert only a single new data record at the lowest level. Because the engine inserts at least one record for any addition, and only one record is called for here, there is no question about how many records to insert, that is, the insert is not ambiguous.

However, if any data values changed in records above the lowest record in the path, an ambiguous situation occurs. A specific number of new records seem to be required by your changes, but some of the new data might already exist in the database records. That is, the actual number of new records to be added to the database might be different.

In insert mode, the engine can determine whether some of the new data already exists in a record. If the data exists, the engine needs to insert records only for the data that does not exist in the database. If the data does not exist, the engine needs to insert a record at every level.

In optimized load mode, the engine ignores the ambiguity; it inserts all of the new data that is at or below the highest-level record that changed. Therefore, when you specify optimized load mode, make sure that your incoming data is always sorted by major-to-minor sort keys at every level (from level 0 down to the lowest level in the view). If the data is not sorted correctly, redundancy occurs.

If you specify a BY key, it should contain one or more database items at each level above the lowest level in your view descriptor.

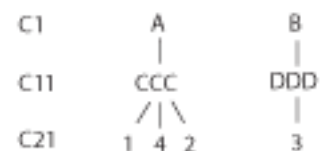
BY keys cause extra processing time because the engine issues one or more where-clauses to look for already-existing records.

Examples Using a BY Key

You have a view with C1 and C11 in the BY key and three observations.



Suppose you are using the FSEDIT procedure on observation 1, and you issue the DUP command and enter values A, CCC, and 4. This is not an ambiguous insert; a BY key is not required. The changes in values from observation 1 to your new input are confined to the lowest level of the view. Here is the result.



Now, suppose you are using the FSEDIT procedure on observation 1, and you issue an ADD or a DUP command and enter the values B, DDD, and 5 for C1, C11, and C21, respectively. The insert is ambiguous because all the fields in the new observation are different from observation 1. Without a BY key, here is the result:

```

C1      A      B      B
  |      |      |      |
C11     CCC     DDD     DDD
  /|\      |      |
C21    1 4 2    3      5

```

With a BY key, the engine finds the BY key values C1=B and C11=DDD in the database. Here is the result:

```

C1      A      B
  |      |      |
C11     CCC     DDD
  /|\      /|\
C21    1 4 2    3 5

```

BY-Key Effects on Performance

The recommended way to use BY keys is to

- include an item at every level above the lowest level of the view descriptor
- standardize all database updates through the same view or through consistent views.

CAUTION:

The engine does not enforce that a BY key must contain at least one item at every level above the lowest level in the view descriptor. However, if the BY key does not contain enough unique items, it might be inadequate to help the engine. The engine might behave as if there were no BY key.

CAUTION:

The engine does not enforce consistent use of BY keys; one view descriptor might have a BY key and another might not. In this instance, redundant data could be added to the database through the view descriptor that does not have a BY key. Also, some applications that use the QUEST procedure could enter redundant data. PROC QUEST does not call the engine for database updates. If data is added in any way other than through a view descriptor using a BY key, the engine might find several qualified database records that match the incoming data. The engine would select one record that works and use it when inserting the new records, and the incoming data might be attached beneath a different existing record than the one that you expect. To avoid this, make sure that all users who update the database follow the same rules. That is, ensure that all data entry is performed through the interface view engine and that all users use the same view descriptor (or consistent view descriptors).

In addition, the content of a prior observation is important during inserts because the engine compares your new data to it. The prior observation is obvious for SAS procedures that pass through a file sequentially, such as the DBLOAD procedure. However, other SAS procedures can pass randomly through a file, such as the FSEDIT procedure.

When you add observations by using procedures that do not use sequential processing, remember that the prior observation is the last observation that the procedure showed

you. For example, in the FSVIEW procedure, the prior observation is the last observation that the procedure displayed at the bottom of your monitor before your first update.

In some instances, there is no prior observation, such as when you use the DBLOAD procedure. PROC DBLOAD calls the engine to add an observation without any prior retrieval. If this occurs, the engine issues a GET1 ... LAST command for the record at the top of the view and retrieves the last record that was inserted into the database.

Missing Values (Nulls)

Retrieving Nulls

When the interface view engine is reading database records and constructing an observation, it might find that data is missing in the path of the data records that represent the observation.

In a SYSTEM 2000 database,

- missing structure means that the data record at the highest level of the view exists, but some or all of its descendant records do not exist.
- missing values (nulls) means that the values for one or more items in a data record do not exist. Nulls for all item types are represented by binary zeros in the database.

In SAS,

- missing values in character variables are represented by blanks
- missing numeric values are represented by a period (.)

When the interface view engine retrieves a null from the database, it sets the null as a missing value in the corresponding SAS observation. Because SYSTEM 2000 preserves all blanks for TEXT and UNDEFINED values, a value that contains all blanks for one of these item types is interpreted as a missing value by a SAS procedure.

Updating Nulls

The interface view engine supports four types of updates: ADD, UPDATE, DUP, and DELETE.

ADD

adds an observation, which can have nulls. The interface view engine converts a SAS observation into a set of one or more SYSTEM 2000 data records, which comprise the path defined by the view descriptor. Each variable in each record is converted from the SAS internal format to the SYSTEM 2000 format. Even if all variables in a SYSTEM 2000 record have nulls, the record is inserted into the database. That is, the complete path of data records is always inserted; lower-level data records might contain all nulls.

UPDATE

updates an observation in a record with a set of values. The record might contain nulls.

If the observation being updated has no missing structure, each variable is converted from its SAS form into a SYSTEM 2000 form.

If the observation being updated has a missing structure in the database, the records that exist in the path are updated with whatever values have changed since the path was retrieved. Missing structures will be inserted only if the values are not null.

DUP

duplicates the selected observation in the database, which can cause duplication of more than one database record.

DELETE

deletes an observation, which can cause deletion of more than one database record. For more information, see [“Deleting Data Records” on page 122](#).

Nulls in Selection Criteria

SYSTEM 2000 and SAS treat nulls differently when processing where-clause conditions. SYSTEM 2000 assumes that a null is outside the domain of values for an item. Therefore, the only way to qualify a null is by using the FAILS operator. In fact, for any relational operator in an item-to-item condition, SYSTEM 2000 never qualifies a record in which either of the items is null. Even if the condition is $C1* = C2*$ and both items are null, the record does not qualify. For example, if item C2 is null in some data records, the following item-to-item condition never qualifies those records, regardless of the respective values:

```
WHERE C1* > C2*
```

In contrast, SAS assumes that nulls are equal to each other. In SAS, nulls

- for numeric variables are indicated by periods
- for character variables are indicated by blanks

When SAS processes a condition such as $C1 \geq C2$, the qualified records include every record in which C2 is null, regardless of the value of C1. Also, the condition $C1 = C2$ qualifies records that have nulls for both C1 and C2, in addition to records in which C1 and C2 have equal values that are not null.

Because of these different treatments, it is important to know whether SAS or SYSTEM 2000 is processing a where-clause. The where-clause in a view descriptor is never seen by SAS and is processed by SYSTEM 2000. However, the WHERE clause associated with a SAS procedure, the DATA step, or a SELECT statement in the SQL procedure can be processed partly by both SAS and SYSTEM 2000 if individual conditions are meaningful to SYSTEM 2000.

Because missing values are different, a condition in a WHERE clause in SAS that uses the period (.) notation is never seen by SYSTEM 2000. SAS performs the qualification for such conditions. For more information, see [“WHERE Clauses in SAS and where-clauses in SYSTEM 2000” on page 127](#).

WHERE Clauses in SAS and where-clauses in SYSTEM 2000

Overview of WHERE Clauses

In addition to, or instead of including a SYSTEM 2000 where-clause in your view descriptor for selection criteria, you can specify a *WHERE clause* in a SAS program for selection criteria.

Note: Unlike a SYSTEM 2000 where-clause that is stored in a view descriptor, a WHERE clause in SAS is restricted to variables that correspond to items included in the view descriptor. (A SYSTEM 2000 where-clause can reference items that are contained in a view descriptor and items that are contained in the access descriptor that the view descriptor is based on.)

When you specify a WHERE clause, the SAS/ACCESS interface view engine translates the specified conditions into SYSTEM 2000 conditions. If the view descriptor includes a SYSTEM 2000 where-clause, the interface view engine connects the conditions with the Boolean operator AND. By default, the conditions in the WHERE clause in SAS are connected to the end of the view descriptor conditions. For example, if a view descriptor includes the condition

```
sex=female
```

and the WHERE clause condition in SAS translates into

```
position=marketing
```

the resulting selection criteria are

```
sex=female and position=marketing
```

You can control the connection of the translated WHERE clause in SAS and the SYSTEM 2000 where-clause conditions by including a connecting string in a SYSTEM 2000 where-clause that is included in a view descriptor. A connecting string indicates where you want the connection to occur. For example, if you include the following SYSTEM 2000 where-clause in a view descriptor (*SASAND* is a connecting string),

```
*sasand* department=marketing
```

and execute a SAS procedure that includes a WHERE clause that produces the following condition:

```
salary gt 1000
```

The resulting selection criteria are

```
salary gt 1000 and department=marketing
```

For more information and examples, see [“Connecting Strings to Order Conditions” on page 133](#).

Because there are capabilities in the WHERE clause in SAS that are not available in SYSTEM 2000, when the interface view engine translates the WHERE clause conditions in SAS into SYSTEM 2000 conditions, it is possible that the WHERE clause in SAS cannot be totally executed in SYSTEM 2000.

For this possibility, the interface view engine first evaluates the WHERE clause in SAS and determines which conditions SYSTEM 2000 can support. The interface view engine might be able to partially execute the WHERE clause. For example, in the following program:

```
proc print data=vlib.emp1;
  where lastname < 'KAP'
    and payrate > 30 * overtime;
run;
```

the interface view engine translates as much of the WHERE clause as possible, without producing incorrect results or a syntax error in SYSTEM 2000. In this example, SYSTEM 2000 can execute the first condition, but the arithmetic in the second condition is not supported. Therefore, the engine uses **where lastname < 'KAP'** to filter out as many data records as possible to improve performance. The conditions that are not supported are bypassed by the engine, and post-processing (performed automatically by

SAS) will be required after SYSTEM 2000 completes its subsetting. The engine bypasses:

- unacceptable conditions.
- conditions connected by OR to unacceptable conditions.
- conditions that exceed the 1000-byte limit of a SYSTEM 2000 where-clause. If the WHERE clause in SAS exceeds 1000 bytes, the rightmost portion of the clause is bypassed by SYSTEM 2000.

When the interface view engine first examines the WHERE clause in SAS and determines which conditions SYSTEM 2000 can support, the engine has not yet processed the view descriptor where-clause. Later, when the engine processes the view descriptor where-clause, the possibility arises that the combined length of the WHERE clause conditions in SAS that can be supported in SYSTEM 2000 and the view descriptor where-clause conditions might exceed 1000 bytes.

If the engine determines that SYSTEM 2000 completely supports the WHERE clause in SAS, but also determines that the conditions cannot be combined due to the 1000-byte limit, an unrecoverable error occurs. To the SAS procedure or DATA step, it appears as if the first "read" observation failed. You might need to carefully examine the error messages in the log to find out what actually happened.

Note: If there is no SYSTEM 2000 where-clause included in the view descriptor and no WHERE clause specified in the SAS program, the interface view engine issues a default where-clause in the form of WHERE *C_n* EXISTS OR *C_n* FAILS, where *C_n* is a component in the lowest-level record in the view descriptor.

The default where-clause "WHERE *C_n* EXISTS OR *C_n* FAILS" guarantees that the view retrieves 100% of the database defined by that view, but it does cause a complete non-key pass of the database. A more efficient default where-clause can be defined by using the following syntax:

*DEFAULT (WHERE valid-subset where-clause)

As a knowledgeable user of your database, you might be able to define a where-clause using all key components that still guarantee that you retrieve 100% of the database defined by this view.

The *DEFAULT where-clause is validated by SYSTEM 2000 at run time. When you specify a SAS WHERE clause, the *DEFAULT is not used. However, when you open a view that has *DEFAULT specified and do not specify a SAS WHERE clause, *DEFAULT is used to qualify the data. The qualified data is passed to the engine for processing by the application or procedure. If a subsequent SAS WHERE clause is specified, the new WHERE clause is the only qualification that is sent to SYSTEM 2000 for retrieval.

In the following table, assume that C114 is a component in the lowest-level record of a view descriptor.

Table A2.1 Translating SYSTEM 2000 where-clause and WHERE Clauses in SAS

where-clause in SYSTEM 2000 View Descriptor	WHERE Clause in SAS	SYSTEM 2000 Translation	Post- Processing Required?
C1=A	C2=B OR C3>C4+10	(C1=A)	Yes

where-clause in SYSTEM 2000 View Descriptor	WHERE Clause in SAS	SYSTEM 2000 Translation	Post- Processing Required?
C1=A	C2=B & C3>C4+10	(C1=A) & (C2=B)	Yes
C1=A	C2=B OR C3>C4	(C1=A) & (C2=B OR C3*>C4*)	No
C1=A	C2=B & C3	(C1=A) & (C2=B)	Yes
---	---	C114 EXISTS OR C114 FAILS	No
---	C3*20 < C5	C114 EXISTS OR C114 FAILS	Yes
---	C3 = C5	C3* = C5*	No

WHERE Clauses in SAS Translatable to SYSTEM 2000

The next three tables show the interface view engine translations of acceptable WHERE clause conditions in SAS into where-clause conditions in SYSTEM 2000.

Table A2.2 SAS Operators Translated into SYSTEM 2000 Operators

WHERE Clause Operators in SAS	SYSTEM 2000 Operators
=	=
>	>
<	<
<>	!=
>=	>=
<=	<=
IS NULL	FAILS
IS NOT NULL	EXISTS
((
))
AND	AND

WHERE Clause Operators in SAS	SYSTEM 2000 Operators
OR	OR

Table A2.3 Additional SAS Syntax Translations into SYSTEM 2000

WHERE Clause Syntax in SAS	SYSTEM 2000 Translation
C1 BETWEEN 1 AND 3	C1 = 1*3
C1 IN (4,9,14)	C1=4 OR C1=9 OR C1=14
C4 > C5	C4* > C5*
C4 = '02AUG87'D	C4 = 08/02/1987

SYSTEM 2000 can handle a limited subset of WHERE clause pattern matching specified in SAS, under the following conditions:

- The pattern must be less than 100 characters in length.
- The pattern must have a percent sign (%) as the last character.
- Underscores () are permitted only in the beginning position(s).
- The pattern cannot have a percent sign (%) anywhere except in the beginning or in the last position.
- The pattern must have some characters that are not percent signs (%) or underscores ().

Table A2.4 SAS Pattern Syntax Translated to SYSTEM 2000

WHERE Clause Syntax in SAS	SYSTEM 2000 Translation
C1 LIKE %ABC%	C1 CONTAINS ABC
C1 LIKE ABC%	C1 CONTAINS ABC IN 1
C1 LIKE _ABC%	C1 CONTAINS ABC IN 2
C1 LIKE __ABC%	C1 CONTAINS ABC IN 3

WHERE Clauses in SAS Not Translatable to SYSTEM 2000

Here are some (but not all) WHERE clause conditions in SAS that are not accepted in SYSTEM 2000. They are executed automatically by SAS post-processing:

- arithmetic expressions such as the following example:

```
WHERE C1 = C4 * 3
WHERE C4 < -C5
```

- expressions in which a variable or combination of variables assumes a value of 1 or 0 to signify true or false, such as in the following example:

```
WHERE C1
WHERE (C1 = C2) * 20
```

- concatenation of character variables.
- truncated comparison, as in the following example:

```
C1 =: ABC
```

- DATETIME and TIME formats, as in the following example:

```
'12:00'T
'01JAN60:12:00'DT
```

- SOUNDEX.
- HAVING, GROUP BY, and NOT CONTAINS conditions.
- references to nulls indicated by a period (.) for numeric variables or closing quotation marks (") for character variables. Use **WHERE C1 IS NULL**, do not use **WHERE C1 = .** or **' '** to indicate a null. The interface view engine can translate **C1 IS NULL** into **C1 FAILS**.

NOT Operator in SAS and SYSTEM 2000

The WHERE clause NOT operator in SAS and the where-clause NOT operator in SYSTEM 2000 do not function the same way. If you want NOT to have its SAS meaning, put it in the WHERE clause in SAS. If you want NOT to have its SYSTEM 2000 meaning, put it in the view descriptor where-clause in SYSTEM 2000.

If you specify NOT in a WHERE clause in SAS, NOT is transformed by the WHERE clause parser in SAS; the interface view engine never sees the NOT operator.

Table A2.5 Examples of the NOT Operator in SAS

WHERE Clause in SAS	What the Interface View Engine Sees
WH NOT LASTNAME = 'Jones';	WH LASTNAME NE 'Jones';
WH NOT LASTNAME > 'Baker';	WH LASTNAME <= 'Baker';
WH NOT (LASTNAME = JONES AND HIREDATE > '02aug82'd);	WH LASTNAME NE 'Jones' OR HIREDATE <= '02aug82'd;

In SYSTEM 2000, the logical converse of **wh not lastname = 'Jones'**; is **wh lastname ne Jones or lastname fails**. Before any relational operator can find a match for a value, the value must exist. One reason for this is that nulls are not contained in SYSTEM 2000 indexes, and processing an operator such as NE could be expensive if it were not confined to indexed values.

Specifying Selection Criteria

The following guidelines help you determine when to use a WHERE clause in SAS and when to use a SYSTEM 2000 where-clause to specify selection criteria.

Use a SYSTEM 2000 where-clause in your view descriptor when you want to do one of the following actions:

- restrict users of view descriptors to specific subsets of data.
- use SYSTEM 2000 syntax and functionality, such as component names, stored strings, HAS, AT, and the NON-KEY specification.
- qualify using a database item that is not in the view descriptor.
- ensure that nulls (missing values) are treated how SYSTEM 2000 expects. (The SYSTEM 2000 handling of nulls differs from SAS in that SYSTEM 2000 does not treat nulls as equal to other values, including other nulls.)
- use the SYSTEM 2000 functionality of the NOT operator. (The SYSTEM 2000 processing of the NOT operator differs from SAS in that SYSTEM 2000 includes null values in the answer, where SAS might or might not include nulls.)
- prevent users from sequentially passing the entire database. (The DBA can also set the SYSTEM 2000 option to DISABLE FULL PASSES as a way of preventing sequential processing.)

Use a WHERE clause in SAS when the preceding guidelines do not apply, and you have one of the following situations:

- you need more run-time flexibility in subsetting data
- you need to use WHERE clause capabilities in SAS that SYSTEM 2000 does not support, such as arithmetic expressions or truncated comparisons

Connecting Strings to Order Conditions

Using Connecting Strings

The order in which SYSTEM 2000 processes conditions can affect which data records are selected. This is most obvious when you include a SYSTEM 2000 where-clause in a view descriptor, and specify a WHERE clause in a SAS program that uses the view descriptor. By default, the interface view engine connects the translated WHERE clause conditions in SAS to the end of the SYSTEM 2000 where-clause conditions by using the Boolean operator AND.

To affect the order of the connected conditions, you can include a connecting string in a SYSTEM 2000 where-clause to tell the engine how you want to connect the conditions. See the following table:

Table A2.6 Examples of Using Connecting Strings

SYSTEM 2000 where-clause in View Descriptor	WHERE Clause in SAS Program	Connected Conditions
C1 = A	C110 > 27	(C1 = A) & (C110 > 27)
SAS & C1 = A	C110 > 27	(C110 > 27) & C1 = A
C1 = 'A' *ANDSAS*	C110 > 27	C1 = 'A' AND (C110 > 27)

Note: Remember that the interface view engine translates only those WHERE conditions in SAS that it understands.

The following table summarizes the connecting strings that you can specify in a SYSTEM 2000 where-clause that is included in a view descriptor.

Table A2.7 Strings to Specify in SYSTEM 2000 where-clauses

Connecting String	Expands to
SAS	(SAS-conditions)
ANDSAS	AND (SAS-conditions)
SASAND	(SAS-conditions) AND
ANDNK	AND (NK (SAS-conditions))
NKAND	(NK (SAS-conditions)) AND
*ANDAT(<i>n</i>)	AND ((SAS-conditions)AT <i>n</i>)
*ATAND(<i>n</i>)	((SAS-conditions) AT <i>n</i>) AND
*ANDHAS(<i>record</i>)	AND (<i>record</i> HAS (SAS-conditions))
*HASAND(<i>record</i>)	(<i>record</i> HAS (SAS-conditions))AND
*HASSAS(<i>record</i>)	(<i>record</i> HAS (SAS-conditions))
NKSAS	NK (SAS-conditions)
*SASAT(<i>n</i>)	(SAS-conditions)AT <i>n</i>

Syntax for Specifying a Connecting String

You can specify a connecting string in a SYSTEM 2000 where-clause after a keyword or a special character. For example,

```
C1 = A AND *SAS*
```

The following syntax is not acceptable:

```
C1 = A *ANDSAS*
```

however, you can use the preceding syntax if you include a delimiter (special character.) In the following example, the delimiter is a set of single quotation marks:

```
C1 = 'A' *ANDSAS*
```

Omitting a WHERE Clause in SAS

If a view descriptor includes a SYSTEM 2000 where-clause with a connecting string, and you do not execute a WHERE clause in SAS, there is nothing to substitute. For example, suppose you have included the following SYSTEM 2000 where-clause (with the connecting string *SAS*) in a view descriptor:

```
C1 = A AND *SAS*
```

Then, you issue a SAS program specifying a WHERE clause that produces the following SYSTEM 2000 condition:

```
C110 > 27
```

If you do not specify a WHERE clause in the SAS program, the "dangling connector" would result in a SYSTEM 2000 error.

```
C1 = A AND
```

If you want the flexibility of omitting the WHERE clause in SAS, you can use the *ANDSAS* or *SASAND* connecting string. For example,

```
C1 = 'A' *ANDSAS*
```

Then, even if you did not specify a WHERE clause in SAS, there would not be a problem. The result would be:

```
C1 = 'A'
```

Using the OR Operator

You cannot use an OR operator to connect a connecting string to other parts of a view descriptor where-clause. For example, the following view descriptor where-clauses are not acceptable:

```
C1 = A OR *SAS*
C1 = C OR (C1 = A OR C1 = B) *ANDSAS*
```

However, you can use the OR operator as shown in the following example:

```
(C1 = A OR C1 = B) AND *SAS*
```

Using HAS, AT, and NON-KEY

The HAS and AT operators and the NON-KEY specification are available in a SYSTEM 2000 where-clause, but they are not available in a WHERE clause in SAS. By using specific connecting strings, you can make the function of HAS, AT, and NON-KEY more useful in the SYSTEM 2000 where-clause and have the option of omitting the WHERE clause in SAS without introducing errors or unexpected results. See the following table for examples.

Table A2.8 HAS, AT, and NON-KEY in SAS and SYSTEM 2000

SYSTEM 2000 where-clause in View Descriptor	WHERE Clause in SAS	Selection Criteria
C1='A' *ANDNK*	C2=B OR C3=X	C1='A' & (NK C2=B OR NK C3=X)
C1='A' *ANDNK*		C1='A'
C1='A' *ANDHAS(C0)	C21=B & C22=X	C1='A' AND (C0 HAS (C21=B & C22=X))
*ATAND(12) C1=A	C21=B	C21=B AT 12 & C1=A

Stored Strings in SYSTEM 2000

When you include a SYSTEM 2000 where-clause in a view descriptor, you can either use where-clause syntax as explained in “[where-clause in SYSTEM 2000](#)” on page 72, or you can refer to a SYSTEM 2000 stored string. A *stored string* is syntax contained in a SYSTEM 2000 database definition that can be invoked by using the string number or name. Either a complete where-clause or a portion of one can be stored. For example, you can store part of a SYSTEM 2000 where-clause in the database, such as the following example:

```
sex=female
```

If you assign string number C1001 to the string, when you include a where-clause in a view descriptor, you can refer to the string number:

```
department=marketing and *c1001*
```

When the selection criteria are processed by SYSTEM 2000 against the database, here is the result:

```
department=marketing and sex=female
```

However, when the interface view engine confronts the view descriptor where-clause, the engine can check for errors only until it encounters the string reference. The engine cannot access the string definition and therefore cannot expand the string to validate your syntax. Also, the engine cannot check the syntax that follows the string expansion, which means you must be more careful with the where-clause construction. However, the engine appends a WHERE clause in SAS at the end of the view descriptor where-clause if this was not done before the occurrence of a SYSTEM 2000 string reference.

If you specify a stored string in a view descriptor where-clause, follow these rules in the where-clause syntax after the string reference:

- Use only valid SYSTEM 2000 item component names or numbers.
- Enter all keywords and any character values in uppercase.
- Do not use connecting strings.
- Do not use TEXT values that contain significant blanks.

Appendix 3

Example Programs

Using the Example Programs	137
SYSTEM 2000 Database Definition for Database EMPLOYEE	137
Access Descriptors	139
Access Descriptor MYLIB.EMPLOYEE	139
View Descriptors	141
Access Descriptor MYLIB.EMPLOYEE	141
View Descriptor VLIB.EMPBD	141
View Descriptor VLIB.EMPEDUC	141
View Descriptor VLIB.EMPPHON	144
View Descriptor VLIB.EMPPPOS	145
View Descriptor VLIB.EMPSKIL	147
View Descriptor VLIB.EMPVAC	150
SAS Data Files	152
Data File MYDATA.CLASSES	152
Data File V6.BIRTHDY	153
Data File V7.CONSULTING_BIRTHDAYS	153
Data File MYDATA.CORPHON	154
Data File Trans.Banking	155

Using the Example Programs

If you want to run the example programs contained in this section, contact your on-site SAS support personnel for information about accessing the Sample Library files. Also, contact your DBA to be sure the data in the database EMPLOYEE is correct. (The database EMPLOYEE might need to be restored if previous users ran the examples given here, which include deletes and inserts.)

SYSTEM 2000 Database Definition for Database EMPLOYEE

The descriptor files created and used in this documentation are based on the complete database definition for the database EMPLOYEE:

Output A3.1 Database Definition for Database EMPLOYEE

```

SYSTEM RELEASE NUMBER  11.6A
DATA BASE NAME IS      EMPLOYEE
DEFINITION NUMBER      2
DATA BASE CYCLE NUMBER  25
  1* EMPLOYEE NUMBER (INTEGER NUMBER 9999)
  2* LAST NAME (CHAR X(10) WITH FEW FUTURE OCCURRENCES )
  3* FORENAME (NON-KEY CHAR X(20))
  4* HIRE DATE (DATE)
  5* BIRTHDAY (DATE)
  6* SOCIAL SECURITY NUMBER (NON-KEY CHAR X(11))
  7* SEX (CHAR X(6) WITH MANY FUTURE OCCURRENCES )
  8* ETHNIC ORIGIN (CHAR X(9) WITH SOME FUTURE OCCURRENCES )
  9* EMPLOYEE STATUS (CHAR X(9) WITH MANY FUTURE OCCURRENCES )
 10* OFFICE-EXTENSION (NON-KEY CHAR X(9))
 11* ACCRUED VACATION (NON-KEY DECIMAL NUMBER 999.99)
 12* ACCRUED SICK LEAVE (NON-KEY DECIMAL NUMBER 999.99)
 13* SECURITY CLEARANCE (INTEGER NUMBER 999 WITH MANY FUTURE OCCURRENCES )
 14* STREET ADDRESS (NON-KEY CHAR X(20))
 15* CITY-STATE (NON-KEY CHAR X(15))
 16* ZIP CODE (CHAR X(5) WITH FEW FUTURE OCCURRENCES )
100* POSITION WITHIN COMPANY (RECORD)
  101* POSITION TITLE (NON-KEY CHAR X(10) IN 100)
  102* DEPARTMENT (CHAR X(14) IN 100 WITH SOME FUTURE OCCURRENCES )
  103* MANAGER (CHAR XXX IN 100 WITH FEW FUTURE OCCURRENCES )
  104* POSITION TYPE (CHAR X(12) IN 100 WITH SOME FUTURE OCCURRENCES )
  105* START DATE (DATE IN 100)
  106* END DATE (NON-KEY DATE IN 100)
 110* SALARY WITHIN POSITION (RECORD IN 100)
    111* PAY RATE (MONEY $9999.99 IN 110)
    112* PAY SCHEDULE (CHAR X(7) IN 110)
    113* EFFECTIVE DATE (DATE IN 110)
    114* CURRENT DEDUCTION (NON-KEY MONEY $9999.99 IN 110)
  120* MONTHLY PAYROLL ACCOUNTING (RECORD IN 110)
    121* PAYROLL MONTH (DATE IN 120)
    122* REGULAR HOURS (NON-KEY DECIMAL NUMBER 999.99 IN 120)
    123* OVERTIME HOURS (NON-KEY DECIMAL NUMBER 999.99 IN 120)
    124* GROSS PAY (NON-KEY MONEY $9999.99 IN 120)
    125* FEDERAL TAX DEDUCTION (NON-KEY MONEY $9999.99 IN 120)
    126* NET PAY (NON-KEY MONEY $9999.99 IN 120)
 130* ADDITIONAL INFORMATION (RECORD IN 100)
    131* LINE NUMBER (DECIMAL NUMBER 99.9 IN 130)
    132* COMMENT TEXT (NON-KEY TEXT X(7) IN 130)
200* JOB SKILLS (RECORD)
  201* SKILL TYPE (CHAR X(12) IN 200 WITH SOME FUTURE OCCURRENCES )
  202* PROFICIENCY (NON-KEY CHAR X(5) IN 200)
  203* YEARS OF EXPERIENCE (NON-KEY INTEGER NUMBER 99 IN 200)
300* PERSONAL INTERESTS (RECORD)
  301* INTEREST (CHAR X(12) IN 300 WITH FEW FUTURE OCCURRENCES )
  302* AFFILIATION (NON-KEY CHAR X(5) IN 300)
  303* COMMENT (NON-KEY TEXT X(5) IN 300)
400* EDUCATIONAL BACKGROUND (RECORD)
  410* EDUCATION (RECORD IN 400)
    411* SCHOOL (CHAR X(15) IN 410)
    412* DEGREE/CERTIFICATE (CHAR X(7) IN 410 WITH FEW FUTURE OCCURRENCES
)
    413* DATE COMPLETED (DATE IN 410)
    414* MAJOR FIELD (NON-KEY CHAR X(16) IN 410)
    415* MINOR FIELD (NON-KEY CHAR X(12) IN 410)
  420* TRAINING (RECORD IN 400)
    421* SOURCE (NON-KEY CHAR X(12) IN 420)
    422* CLASS NAME (CHAR X(12) IN 420 WITH FEW FUTURE OCCURRENCES )
    423* DATE ACCOMPLISHED (DATE IN 420)

```

Access Descriptors

Access Descriptor MYLIB.EMPLOYEE

The access descriptor MYLIB.EMPLOYEE for the database EMPLOYEE is used in most of the examples in this documentation. You can create the access descriptor MYLIB.EMPLOYEE by using the following program in batch or in interactive line mode.

```
proc access dbms=s2k;
  create mylib.employe.access;
  database=employee
  s2kpw=demo mode=s;
  assign=yes;
  rename forename=firstnme office_e=phone
         yearsofe=years gender=sex
         degree_c=degree;
  length firstnme=13 lastname=13 c101=16;
  list all;
run;
```

The following results are printed to the SAS log:

Output A3.2 Listing of Access Descriptor MYLIB.EMPLOYEE

```

SYSTEM 2000 Database: EMPLOYEE
Function: CREATE Descriptors- access: EMPLOYEE view:

```

Item	C-num	SAS Name	Len	Format	Informat	BY-key
1	C0	*RECORD*		*RECORD*	*RECORD*	
2	C1	EMPLOYEE		4.0	4.0	
3	C2	LASTNAME	13	\$13.	\$13.	
4	C3	FIRSTNME	13	\$13.	\$13.	
5	C4	HIREDATE		DATE7.	DATE7.	
6	C5	BIRTHDAY		DATE7.	DATE7.	
7	C6	SOCIALSE		\$11.	\$11.	
8	C7	SEX		\$6.	\$6.	
9	C8	ETHNICOR		\$9.	\$9.	
10	C9	EMPLOYEE0		\$9.	\$9.	
11	C10	PHONE		\$9.	\$9.	
12	C11	ACCRUEDV		6.2	6.2	
13	C12	ACCRUEDS		6.2	6.2	
14	C13	SECURITY		3.0	3.0	
15	C14	STREETAD		\$20.	\$20.	
16	C15	CITY_STA		\$15.	\$15.	
17	C16	ZIPCODE		\$5.	\$5.	
18	C100	*RECORD*		*RECORD*	*RECORD*	
19	C101	POSITION		\$10.	\$10.	
20	C102	DEPARTME		\$14.	\$14.	
21	C103	MANAGER		\$3.	\$3.	
22	C104	POSITIO1		\$12.	\$12.	
23	C105	STARTDAT		DATE7.	DATE7.	
24	C106	ENDDATE		DATE7.	DATE7.	
25	C110	*RECORD*		*RECORD*	*RECORD*	
26	C111	PAYRATE		7.2	7.2	
27	C112	PAYSCHED		\$7.	\$7.	
28	C113	EFFECTIV		DATE7.	DATE7.	
29	C114	CURRENTD		7.2	7.2	
30	C120	*RECORD*		*RECORD*	*RECORD*	
31	C121	PAYROLLM		DATE7.	DATE7.	
32	C122	REGULARH		6.2	6.2	
33	C123	OVERTIME		6.2	6.2	
34	C124	GROSSPAY		7.2	7.2	
35	C125	FEDERALT		7.2	7.2	
36	C126	NETPAY		7.2	7.2	
37	C130	*RECORD*		*RECORD*	*RECORD*	
38	C131	LINENUMB		4.1	4.1	
39	C132	COMMENTT		\$CHAR7.	\$CHAR7.	
40	C200	*RECORD*		*RECORD*	*RECORD*	
41	C201	SKILLTYP		\$12.	\$12.	
42	C202	PROFICIE		\$5.	\$5.	
43	C203	YEARSOFE		2.0	2.0	
44	C300	*RECORD*		*RECORD*	*RECORD*	
45	C301	INTEREST		\$12.	\$12.	
46	C302	AFFILIAT		\$5.	\$5.	
47	C303	COMMENT		\$CHAR5.	\$CHAR5.	
48	C400	*RECORD*		*RECORD*	*RECORD*	
49	C410	*RECORD*		*RECORD*	*RECORD*	
50	C411	SCHOOL		\$15.	\$15.	
51	C412	DEGREE		\$7.	\$7.	
52	C413	DATECOMP		DATE7.	DATE7.	
53	C414	MAJORFIE		\$16.	\$16.	
54	C415	MINORFIE		\$12.	\$12.	
55	C420	*RECORD*		*RECORD*	*RECORD*	
56	C421	SOURCE		\$12.	\$12.	
57	C422	CLASSNAM		\$12.	\$12.	
58	C423	DATEACCO		DATE7.	DATE7.	

View Descriptors

Access Descriptor MYLIB.EMPLOYEE

You can create all the view descriptors used in this documentation by using the following SAS programs in batch or in interactive line mode. All the view descriptors are based on the access descriptor MYLIB.EMPLOYEE.

View Descriptor VLIB.EMPBD

The view descriptor VLIB.EMPBD was created by using the following program. This view descriptor accesses the data shown in the output below the program.

```
proc access dbms=s2k ad=mylib.employe;
  create vlib.empbd.view;
  select lastname firstnme birthday;
  subset "ob lastname,firstnme";
  s2kpw=demo mode=s;
  list view;
run;
```

Output A3.3 Data Accessed by VLIB.EMPBD

Data Accessed by VLIB.EMPBD				1
OBS	LASTNAME	FIRSTNME	BIRTHDAY	
1	AMEER	DAVID	10OCT51	
2	BROOKS	RUBEN R.	25FEB52	
3	BROWN	VIRGINA P.	24MAY46	
4	CHAN	TAI	04JUL46	
5	GARRETT	OLAN M.	23JAN35	
6	GIBSON	GEORGE J.	23APR46	
7	GOODSON	ALAN F.	21JUN50	
8	JUAREZ	ARMANDO	28MAY47	
9	LITTLEJOHN	FANNIE	17MAY54	
10	RICHARDSON	TRAVIS Z.	30NOV37	
11	RODRIGUEZ	ROMUALDO R	09FEB29	
12	SCHOLL	MADISON A.	19MAR45	
13	SHROPSHIRE	LELAND G.	04SEP49	
14	SMITH	JERRY LEE	13SEP42	
15	VAN HOTTEN	GWENDOLYN	13SEP42	
16	WAGGONNER	MERRILEE D	27APR36	
17	WILLIAMSON	JANICE L.	19MAY52	

View Descriptor VLIB.EMPEDUC

The view descriptor VLIB.EMPEDUC was created using the following program. This view descriptor accesses the data shown in the output below the program.

```
proc access dbms=s2k ad=mylib.employe;
  create vlib.empeduc.view;
  select lastname firstnme sex degree;
```

```
subset "ob lastname,firstnme";  
s2kpw=demo mode=s;  
list view;  
run;
```

Output A3.4 Data Accessed by VLIB.EMPEDUC

Data Accessed by VLIB.EMPEDUC					1
OBS	LASTNAME	FIRSTNME	SEX	DEGREE	
1					
2	AMEER	DAVID	MALE	BS	
3	BOWMAN	HUGH E.	MALE	MS	
4	BOWMAN	HUGH E.	MALE	PHD	
5	BOWMAN	HUGH E.	MALE	BS	
6	BROOKS	RUBEN R.	MALE	BS	
7	BROWN	VIRGINA P.	FEMALE	BA	
8	CAHILL	JACOB	MALE	BS	
9	CAHILL	JACOB	MALE	BS	
10	CANADY	FRANK A.	MALE	MA	
11	CANADY	FRANK A.	MALE	BS	
12	CHAN	TAI	MALE	PHD	
13	CHAN	TAI	MALE	BA	
14	COLLINS	LILLIAN	FEMALE	HIGH SC	
15	FAULKNER	CARRIE ANN	FEMALE		
16	FERNANDEZ	SOPHIA	FEMALE	BS	
17	FERNANDEZ	SOPHIA	FEMALE	MS	
18	FREEMAN	LEOPOLD	MALE	BS	
19	FREEMAN	LEOPOLD	MALE	BS	
20	GARCIA	FRANCISCO	MALE	MBA	
21	GARCIA	FRANCISCO	MALE	BS	
22	GARRETT	OLAN M.	MALE	MS	
23	GARRETT	OLAN M.	MALE	BS	
24	GIBSON	MOLLY I.	FEMALE	BA	
25	GIBSON	GEORGE J.	MALE	BA	
26	GIBSON	GEORGE J.	MALE	MS	
27	GIBSON	GEORGE J.	MALE	MS	
28	GOODSON	ALAN F.	MALE	BA	
29	HERNANDEZ	JESSE L.	MALE	PHD	
30	HERNANDEZ	JESSE L.	MALE	MA	
31	HERNANDEZ	JESSE L.	MALE	BS	
32	HERNANDEZ	JESSE L.	MALE	BA	
33	JOHNSON	BRADFORD	MALE		
34	JONES	MICHAEL Y.	MALE	BS	
35	JONES	MICHAEL Y.	MALE		
36	JONES	RITA M.	FEMALE		
37	JUAREZ	ARMANDO	MALE	MS	
38	JUAREZ	ARMANDO	MALE	BS	
39	KAATZ	FREDDIE	MALE	HIGH SC	
40	KNAPP	PATRICE R.	FEMALE	BA	
41	KNIGHT	ALTHEA	FEMALE		
42	LITTLEJOHN	FANNIE	FEMALE	HIGH SC	
43	MILLSAP	JOEL B.	MALE	PHD	
44	MUELLER	PATSY	FEMALE	HIGH SC	
45	NATHANIEL	DARRYL	MALE	AA	
46	POLANSKI	IVAN L.	MALE	BS	
47	POLANSKI	IVAN L.	MALE	BS	
48	POLANSKI	IVAN L.	MALE	MS	
49	QUINTERO	PEDRO	MALE	BS	
50	QUINTERO	PEDRO	MALE		
51	REDFOX	RICHARD B.	MALE	BS	
52	REED	KENNETH D.	MALE		
53	REID	DAVID G.	MALE	BS	
54	RICHARDSON	TRAVIS Z.	MALE	BS	
55	RODRIGUEZ	ROMUALDO R	MALE	BS	

(continued)

56	SALAZAR	YOLANDA	FEMALE	AA
57	SAVAGE	WILLIAM D.	MALE	
58	SCHMIDT	PENNY	FEMALE	HIGH SC
59	SCHOLL	MADISON A.	MALE	MS
60	SCHOLL	MADISON A.	MALE	BS
61	SEATON	GARY	MALE	
62	SHROPSHIRE	LELAND G.	MALE	BA
63	SLYE	LEONARD R.	MALE	HIGH SC
64	SMITH	JERRY LEE	MALE	MA
65	SMITH	JERRY LEE	MALE	BA
66	SMITH	GARLAND P.	MALE	AA
67	SMITH	JANET F.	FEMALE	BA
68	THROCKMORT	STEWART Q.	MALE	BS
69	THROCKMORT	STEWART Q.	MALE	MS
70	VAN HOTTEN	GWENDOLYN	FEMALE	BA
71	WAGGONNER	MERRILEE D	FEMALE	AA
72	WATERHOUSE	CLIFTON P.	MALE	
73	WATERHOUSE	CLIFTON P.	MALE	
74	WILLIAMSON	JANICE L.	FEMALE	BA
75	WILLIAMSON	JANICE L.	FEMALE	AA

View Descriptor *VLIB.EMPPHON*

The view descriptor *VLIB.EMPPHON* was created by using the following program. This view descriptor accesses the data shown in the output below the program.

```
proc access dbms=s2k ad=mylib.employe;
  create vlib.empphon.view;
  select lastname firstnme phone;
  subset "ob lastname,firstnme";
  s2kpw=demo mode=s;
  list view;
run;
```

Output A3.5 Data Accessed by *VLIB.EMPPHON*

Data Accessed by <i>VLIB.EMPPHON</i>				1
OBS	LASTNAME	FIRSTNME	PHONE	
1	AMEER	DAVID	545 XT495	
2	BROOKS	RUBEN R.	581 XT347	
3	BROWN	VIRGINA P.	218 XT258	
4	CHAN	TAI	292 XT331	
5	GARRETT	OLAN M.	212 XT208	
6	GIBSON	GEORGE J.	327 XT703	
7	GOODSON	ALAN F.	323 XT512	
8	JUAREZ	ARMANDO	506 XT987	
9	LITTLEJOHN	FANNIE	219 XT653	
10	RICHARDSON	TRAVIS Z.	243 XT325	
11	RODRIGUEZ	ROMUALDO R	243 XT874	
12	SCHOLL	MADISON A.	318 XT419	
13	SHROPSHIRE	LELAND G.	327 XT616	
14	SMITH	JERRY LEE	327 XT169	
15	VAN HOTTEN	GWENDOLYN	212 XT311	
16	WAGGONNER	MERRILEE D	244 XT914	
17	WILLIAMSON	JANICE L.	218 XT802	

View Descriptor VLIB.EMPPOS

The view descriptor VLIB.EMPPOS was created by using the following program. This view descriptor accesses the data shown in the output following the program.

```
proc access dbms=s2k ad=mylib.employe;
  create vlib.emppos.view;
  select lastname firstnme position departme
         manager;
  subset "order by lastname";
  s2kpw=demo mode=s;
  list all;
run;
```

Output A3.6 Data Accessed by VLIB.EMPPOS

Data Accessed by VLIB.EMPPOS						1
OBS	LASTNAME	FIRSTNME	POSITION	DEPARTME	MANAGER	
1			PROGRAMMER	INFORMATION SY	MYJ	
2	AMEER	DAVID	SR SALES REPRESE	MARKETING	VPB	
3	AMEER	DAVID	JR SALES REPRESE	MARKETING	VPB	
4	BOWMAN	HUGH E.	EXECUTIVE VICE-P	CORPORATION	CPW	
5	BROOKS	RUBEN R.	JR SALES REPRESE	MARKETING	MAS	
6	BROWN	VIRGINA P.	MANAGER WESTERN	MARKETING	OMG	
7	CAHILL	JACOB	MANAGER SYSTEMS	INFORMATION SY	JBM	
8	CANADY	FRANK A.	MANAGER PERSONNE	ADMINISTRATION	PRK	
9	CHAN	TAI	SR SALES REPRESE	MARKETING	TZR	
10	COLLINS	LILLIAN	MAIL CLERK	ADMINISTRATION	SQT	
11	FAULKNER	CARRIE ANN	SECRETARY	CORPORATION	JBM	
12	FERNANDEZ	SOPHIA	STANDARDS & PROC	INFORMATION SY	JLH	
13	FREEMAN	LEOPOLD	SR SYSTEMS PROGR	INFORMATION SY	JLH	
14	GARCIA	FRANCISCO	JR PROGRAMMER/AN	INFORMATION SY	MYJ	
15	GARRETT	OLAN M.	SR SALES REPRESE	MARKETING	VPB	
16	GARRETT	OLAN M.	MANAGER OF SALES	MARKETING	HEB	
17	GARRETT	OLAN M.	VICE-PRESIDENT M	CORPORATION	HEB	
18	GIBSON	MOLLY I.	TECHNICAL WRITER	INFORMATION SY	JC	
19	GIBSON	GEORGE J.	INSTRUCTOR	MARKETING	GVH	
20	GOODSON	ALAN F.	SR SALES REPRESE	MARKETING	TZR	
21	HERNANDEZ	JESSE L.	MANAGER DATA BAS	INFORMATION SY	JBM	
22	JOHNSON	BRADFORD	JR SYSTEMS PROGR	INFORMATION SY	JFS	
23	JONES	RITA M.	MANAGER ACCOUNTI	ADMINISTRATION	PRK	
24	JONES	MICHAEL Y.	SR SYSTEMS ANALY	INFORMATION SY	JC	
25	JUAREZ	ARMANDO	SR SALES REPRESE	MARKETING	VPB	
26	JUAREZ	ARMANDO	JR SALES REPRESE	MARKETING	VPB	
27	KAATZ	FREDDIE	SUPPLY CLERK	ADMINISTRATION	SQT	
28	KNAPP	PATRICE R.	VICE-PRESIDENT A	CORPORATION	HEB	
29	KNIGHT	ALTHEA	SECRETARY	CORPORATION	OMG	
30	LITTLEJOHN	FANNIE	SECRETARY	MARKETING	VPB	
31	MILLSAP	JOEL B.	VICE-PRESIDENT I	CORPORATION	HEB	
32	MUELLER	PATSY	SECRETARY	CORPORATION	PRK	
33	NATHANIEL	DARRYL	SECRETARY	CORPORATION	HEB	
34	POLANSKI	IVAN L.	MANAGER SYSTEMS	INFORMATION SY	JBM	
35	QUINTERO	PEDRO	OPERATIONS SUPER	INFORMATION SY	ILP	
36	REDFOX	RICHARD B.	SYSTEMS ANALYST	INFORMATION SY	JC	
37	REED	KENNETH D.	COMPUTER LIBRARI	INFORMATION SY	PQ	
38	REID	DAVID G.	PROGRAMMER	INFORMATION SY	MYJ	
39	REID	DAVID G.	ASSISTANT PROGRA	INFORMATION SY	MYJ	
40	RICHARDSON	TRAVIS Z.	MANAGER EASTERN	MARKETING	OMG	

(continued)

41	RODRIGUEZ	ROMUALDO R.	PR & ADVERTISING	MARKETING	GVH
42	SALAZAR	YOLANDA	ADMINISTRATIVE A	CORPORATION	CPW
43	SALAZAR	YOLANDA	SECRETARY	CORPORATION	CPW
44	SAVAGE	WILLIAM D.	COMPUTER OPERATO	INFORMATION SY	PQ
45	SCHMIDT	PENNY	SECRETARY	ADMINISTRATION	FAC
46	SCHOLL	MADISON A.	JR SALES REPRESE	MARKETING	VPB
47	SCHOLL	MADISON A.	SR SALES REPRESE	MARKETING	VPB
48	SEATON	GARY	COMPUTER OPERATO	INFORMATION SY	PQ
49	SHROPSHIRE	LELAND G.	JR SALES REPRESE	MARKETING	TZR
50	SLYE	LEONARD R.	GENERAL MAINTENA	ADMINISTRATION	SQT
51	SMITH	JANET F.	SR SYSTEMS PROGR	INFORMATION SY	ILP
52	SMITH	JERRY LEE	JR SALES REPRESE	MARKETING	AFG
53	SMITH	GARLAND P.	BOOKKEEPER	ADMINISTRATION	RMJ
54	THROCKMORT	STEWART Q.	OFFICE SUPERVISO	ADMINISTRATION	FAC
55	VAN HOTTEN	GWENDOLYN	MANAGER PUBLIC R	MARKETING	OMG
56	WAGGONNER	MERRILEE D.	SECRETARY	MARKETING	TZR
57	WATERHOUSE	CLIFTON P.	PRESIDENT	CORPORATION	
58	WILLIAMSON	JANICE L.	INSTRUCTOR	MARKETING	GVH

View Descriptor VLIB.EMPSKIL

The view descriptor VLIB.EMPSKIL was created using the following program. This view descriptor accesses the data shown in the output following the program.

```
proc access dbms=s2k ad=mylib.employe;
  create vlib.empskil.view;
  select c2 c3 c201 c203;
  subset "ob skilltyp";
  s2kpw=demo mode=multi;
  list view;
run;
```

Output A3.7 Data Accessed by VLIB.EMPSKIL

Data Accessed by VLIB.EMPSKIL					1
OBS	LASTNAME	FIRSTNME	SKILLTYP	YEARS	
1				.	
2	AMEER	DAVID	PASCAL	3	
3	BOWMAN	HUGH E.	DP SYSTEMS A	11	
4	BOWMAN	HUGH E.	RUSSIAN	15	
5	BOWMAN	HUGH E.	TEACHING	7	
6	BOWMAN	HUGH E.	PUBLIC RELAT	13	
7	BROOKS	RUBEN R.	PASCAL	4	
8	BROOKS	RUBEN R.	PUBLIC RELAT	1	
9	BROWN	VIRGINA P.	SYSTEMS PROG	2	
10	BROWN	VIRGINA P.	PUBLIC RELAT	8	
11	BROWN	VIRGINA P.	ASSEMBLER	3	
12	BROWN	VIRGINA P.	FRENCH	11	
13	CAHILL	JACOB	TECHNICAL WR	10	
14	CAHILL	JACOB	ASSEMBLER	11	
15	CAHILL	JACOB	COBOL	11	
16	CAHILL	JACOB	SYSTEMS PROG	16	
17	CANADY	FRANK A.	ACCOUNTING	15	
18	CANADY	FRANK A.	COBOL	4	
19	CANADY	FRANK A.	TYPING	20	
20	CHAN	TAI	SYSTEMS PROG	6	
21	CHAN	TAI	CHINESE	8	
22	COLLINS	LILLIAN	TYPING	2	
23	COLLINS	LILLIAN	SHORTHAND	1	
24	FAULKNER	CARRIE ANN	GRAPHICS	1	
25	FAULKNER	CARRIE ANN	SHORTHAND	5	
26	FAULKNER	CARRIE ANN	TYPING	6	
27	FERNANDEZ	SOPHIA	SYSTEMS PROG	6	
28	FERNANDEZ	SOPHIA	DATA BASE AD	3	
29	FERNANDEZ	SOPHIA	SYSTEMS ANAL	12	
30	FERNANDEZ	SOPHIA	PL/1	2	
31	FERNANDEZ	SOPHIA	PASCAL	1	
32	FERNANDEZ	SOPHIA	COBOL	4	

(continued)

33	FERNANDEZ	SOPHIA	FORTTRAN	5
34	FERNANDEZ	SOPHIA	ASSEMBLER	8
35	FREEMAN	LEOPOLD	COBOL	20
36	FREEMAN	LEOPOLD	JAPANESE	3
37	FREEMAN	LEOPOLD	SYSTEMS PROG	20
38	FREEMAN	LEOPOLD	DATA BASE AD	6
39	FREEMAN	LEOPOLD	ASSEMBLER	20
40	GARCIA	FRANCISCO	COBOL	3
41	GARCIA	FRANCISCO	PASCAL	3
42	GARCIA	FRANCISCO	FORTTRAN	3
43	GARRETT	OLAN M.	PUBLIC SPEAK	20
44	GARRETT	OLAN M.	PUBLIC RELAT	8
45	GARRETT	OLAN M.	WRITING	13
46	GARRETT	OLAN M.	FINANCIAL AU	7
47	GIBSON	MOLLY I.	GRAPHICS	3
48	GIBSON	MOLLY I.	CARTOON ART	1
49	GIBSON	GEORGE J.	PUBLIC SPEAK	15
50	GIBSON	GEORGE J.	TEACHING	10
51	GIBSON	GEORGE J.	SYSTEMS ANAL	8
52	GOODSON	ALAN F.	PASCAL	1
53	HERNANDEZ	JESSE L.	ASSEMBLER	24
54	HERNANDEZ	JESSE L.	SYSTEMS PROG	25
55	HERNANDEZ	JESSE L.	SYSTEMS ANAL	17
56	HERNANDEZ	JESSE L.	DATA BASE AD	10
57	HERNANDEZ	JESSE L.	FORTTRAN	19
58	HERNANDEZ	JESSE L.	SPANISH	46
59	JOHNSON	BRADFORD	SYSTEMS PROG	3
60	JONES	MICHAEL Y.	PL/1	10
61	JONES	MICHAEL Y.	ASSEMBLER	17
62	JONES	MICHAEL Y.	COBOL	21
63	JONES	MICHAEL Y.	SYSTEMS PROG	11
64	JONES	MICHAEL Y.	PASCAL	3
65	JONES	MICHAEL Y.	GERMAN	7
66	JONES	MICHAEL Y.	TECHNICAL WR	4
67	JONES	RITA M.	.	.
68	JUAREZ	ARMANDO	ASSEMBLER	7
69	KAATZ	FREDDIE	.	.
70	KNAPP	PATRICE R.	CPA	14
71	KNAPP	PATRICE R.	WRITING	3
72	KNIGHT	ALTHEA	GERMAN	3
73	KNIGHT	ALTHEA	ACCOUNTING	2
74	KNIGHT	ALTHEA	SHORTHAND	8
75	KNIGHT	ALTHEA	TYPING	8
76	KNIGHT	ALTHEA	ETS OPERATOR	1
77	LITTLEJOHN	FANNIE	SHORTHAND	4
78	LITTLEJOHN	FANNIE	GRAPHICS	3
79	LITTLEJOHN	FANNIE	TYPING	4
80	MILLSAP	JOEL B.	SYSTEMS ANAL	17
81	MILLSAP	JOEL B.	HEBREW	4
82	MILLSAP	JOEL B.	SYSTEMS PROG	15
83	MILLSAP	JOEL B.	FORTTRAN	13
84	MILLSAP	JOEL B.	ASSEMBLER	14
85	MUELLER	PATSY	TYPING	6
86	MUELLER	PATSY	SHORTHAND	5
87	MUELLER	PATSY	ACCOUNTING	5
88	NATHANIEL	DARRYL	ACCOUNTING	9
89	NATHANIEL	DARRYL	TYPING	9
90	NATHANIEL	DARRYL	SHORTHAND	7
91	NATHANIEL	DARRYL	PUBLIC RELAT	3

(continued)

92	POLANSKI	IVAN L.	RUSSIAN	3
93	POLANSKI	IVAN L.	ASSEMBLER	7
94	POLANSKI	IVAN L.	OPERATIONS R	10
95	POLANSKI	IVAN L.	SYSTEMS PROG	10
96	QUINTERO	PEDRO	OPERATIONS R	4
97	QUINTERO	PEDRO	SYSTEMS PROG	5
98	QUINTERO	PEDRO	ASSEMBLER	3
99	REDFOX	RICHARD B.	SYSTEMS PROG	10
100	REDFOX	RICHARD B.	ASSEMBLER	15
101	REDFOX	RICHARD B.	SYSTEMS ANAL	12
102	REDFOX	RICHARD B.	FORTRAN	9
103	REED	KENNETH D.	LIBRARY SCIE	1
104	REID	DAVID G.	FORTRAN	1
105	REID	DAVID G.	PL/1	5
106	REID	DAVID G.	COBOL	1
107	RICHARDSON	TRAVIS Z.	JAPANESE	7
108	RODRIGUEZ	ROMUALDO R	GRAPHICS	20
109	RODRIGUEZ	ROMUALDO R	VISUAL DESIG	22
110	SALAZAR	YOLANDA	SHORTHAND	10
111	SALAZAR	YOLANDA	ACCOUNTING	4
112	SALAZAR	YOLANDA	TYPING	13
113	SAVAGE	WILLIAM D.	COBOL	4
114	SCHMIDT	PENNY	SHORTHAND	12
115	SCHMIDT	PENNY	TYPING	23
116	SCHMIDT	PENNY	ETS OPERATOR	3
117	SCHOLL	MADISON A.	SYSTEMS ANAL	9
118	SCHOLL	MADISON A.	FORTRAN	7
119	SEATON	GARY	.	.
120	SHROPSHIRE	LELAND G.	PUBLIC SPEAK	6
121	SLYE	LEONARD R.	PRINT SHOP	0
122	SMITH	JERRY LEE	SYSTEMS DESI	3
123	SMITH	JANET F.	SYSTEMS PROG	8
124	SMITH	JANET F.	ASSEMBLER	7
125	SMITH	JANET F.	COBOL	4
126	SMITH	GARLAND P.	.	.
127	THROCKMORT	STEWART Q.	.	.
128	VAN HOTTEN	GWENDOLYN	TYPING	12
129	VAN HOTTEN	GWENDOLYN	GRAPHICS	8
130	VAN HOTTEN	GWENDOLYN	TECHNICAL WR	4
131	VAN HOTTEN	GWENDOLYN	SHORTHAND	1
132	VAN HOTTEN	GWENDOLYN	PUBLIC SPEAK	9
133	WAGGONNER	MERRILEE D	TYPING	19
134	WAGGONNER	MERRILEE D	SHORTHAND	15
135	WATERHOUSE	CLIFTON P.	COBOL	10
136	WATERHOUSE	CLIFTON P.	FRENCH	21
137	WATERHOUSE	CLIFTON P.	PUBLIC RELAT	19
138	WATERHOUSE	CLIFTON P.	SYSTEMS ANAL	21
139	WATERHOUSE	CLIFTON P.	PUBLIC SPEAK	12
140	WATERHOUSE	CLIFTON P.	ACCOUNTING	12
141	WATERHOUSE	CLIFTON P.	ASSEMBLER	4
142	WILLIAMSON	JANICE L.	COBOL	2
143	WILLIAMSON	JANICE L.	ASSEMBLER	1
144	WILLIAMSON	JANICE L.	COBOL	4
145	WILLIAMSON	JANICE L.	TECHNICAL WR	2
146	WILLIAMSON	JANICE L.	TEACHING	1

View Descriptor *VLIB.EMPVAC*

The view descriptor *VLIB.EMPVAC* was created by using the following program. This view descriptor accesses the data shown in the output following the program.

```
proc access dbms=s2k ad=mylib.employe;
  create vlib.empvac.view;
```

```

select lastname firstnme accruedv departme;
subset "ob lastname,firstnme";
s2kpw=demo mode=s;
list view;
run;

```

Output A3.8 Data Accessed by VLIB.EMPVAC

Data Accessed by VLIB.EMPVAC					1
OBS	LASTNAME	FIRSTNME	ACCRUEDV	DEPARTME	
1				INFORMATION SY	
2	AMEER	DAVID	56.00	MARKETING	
3	AMEER	DAVID	56.00	MARKETING	
4	BOWMAN	HUGH E.	40.00	CORPORATION	
5	BROOKS	RUBEN R.	80.00	MARKETING	
6	BROWN	VIRGINA P.	48.00	MARKETING	
7	CAHILL	JACOB	60.00	INFORMATION SY	
8	CANADY	FRANK A.	8.00	ADMINISTRATION	
9	CHAN	TAI	40.00	MARKETING	
10	COLLINS	LILLIAN	80.00	ADMINISTRATION	
11	FAULKNER	CARRIE ANN	48.00	CORPORATION	
12	FERNANDEZ	SOPHIA	96.00	INFORMATION SY	
13	FREEMAN	LEOPOLD	.	INFORMATION SY	
14	GARCIA	FRANCISCO	80.00	INFORMATION SY	
15	GARRETT	OLAN M.	80.00	MARKETING	
16	GARRETT	OLAN M.	80.00	CORPORATION	
17	GARRETT	OLAN M.	80.00	MARKETING	
18	GIBSON	GEORGE J.	80.00	MARKETING	
19	GIBSON	MOLLY I.	40.00	INFORMATION SY	
20	GOODSON	ALAN F.	48.00	MARKETING	
21	HERNANDEZ	JESSE L.	56.00	INFORMATION SY	
22	JOHNSON	BRADFORD	40.00	INFORMATION SY	
23	JONES	RITA M.	24.00	ADMINISTRATION	
24	JONES	MICHAEL Y.	80.00	INFORMATION SY	
25	JUAREZ	ARMANDO	48.00	MARKETING	
26	JUAREZ	ARMANDO	48.00	MARKETING	
27	KAATZ	FREDDIE	80.00	ADMINISTRATION	
28	KNAPP	PATRICE R.	8.00	CORPORATION	
29	KNIGHT	ALTHEA	0.00	CORPORATION	
30	LITTLEJOHN	FANNIE	8.00	MARKETING	
31	MILLSAP	JOEL B.	24.00	CORPORATION	
32	MUELLER	PATSY	40.00	CORPORATION	
33	NATHANIEL	DARRYL	40.00	CORPORATION	
34	POLANSKI	IVAN L.	56.00	INFORMATION SY	
35	QUINTERO	PEDRO	32.00	INFORMATION SY	
36	REDFOX	RICHARD B.	48.00	INFORMATION SY	
37	REED	KENNETH D.	64.00	INFORMATION SY	
38	REID	DAVID G.	80.00	INFORMATION SY	
39	REID	DAVID G.	80.00	INFORMATION SY	
40	RICHARDSON	TRAVIS Z.	88.00	MARKETING	
41	RODRIGUEZ	ROMUALDO R	32.00	MARKETING	
42	SALAZAR	YOLANDA	80.00	CORPORATION	

(continued)

43	SALAZAR	YOLANDA	80.00	CORPORATION
44	SAVAGE	WILLIAM D.	80.00	INFORMATION SY
45	SCHMIDT	PENNY	80.00	ADMINISTRATION
46	SCHOLL	MADISON A.	40.00	MARKETING
47	SCHOLL	MADISON A.	40.00	MARKETING
48	SEATON	GARY	80.00	INFORMATION SY
49	SHROPSHIRE	LELAND G.	32.00	MARKETING
50	SLYE	LEONARD R.	0.00	ADMINISTRATION
51	SMITH	JANET F.	16.00	INFORMATION SY
52	SMITH	JERRY LEE	0.00	MARKETING
53	SMITH	GARLAND P.	8.00	ADMINISTRATION
54	THROCKMORT	STEWART Q.	64.00	ADMINISTRATION
55	VAN HOTTEN	GWENDOLYN	0.00	MARKETING
56	WAGGONNER	MERRILEE D	56.00	MARKETING
57	WATERHOUSE	CLIFTON P.	8.00	CORPORATION
58	WILLIAMSON	JANICE L.	40.00	MARKETING

SAS Data Files

Data File MYDATA.CLASSES

The SAS data file MYDATA.CLASSES (used in “SYSTEM 2000 Data in SAS Programs”) was created by using the following SAS program:

```
libname mydata 'your-SAS-library';
data mydata.classes;
    input lastname $ 1-10 firstnme $ 15-25 class $ 30-50;
    datalines;
AMMER          DAVID          PRESENTING IDEAS
CANADY         FRANK A.        PRESENTING IDEAS
GIBSON         MOLLY I.        SUPERVISOR SKILLS
GIBSON         MOLLY I.        STRESS MGMT
RICHARDSON     TRAVIS Z.       SUPERVISOR SKILLS
;
```

The output shows the results after running the following program on the data file:

```
proc print data=mydata.classes;
    title2 'SAS Data File MYDATA.CLASSES';
run;
```

Output A3.9 SAS Data File MYDATA.CLASSES

SAS Data File MYDATA.CLASSES				1
OBS	LASTNAME	FIRSTNME	CLASS	
1	AMEER	DAVID	PRESENTING IDEAS	
2	CANADY	FRANK A.	PRESENTING IDEAS	
3	GIBSON	MOLLY I.	SUPERVISOR SKILLS	
4	GIBSON	MOLLY I.	STRESS MGMT	
5	RICHARDSON	TRAVIS Z.	SUPERVISOR SKILLS	

Data File V6.BIRTHDY

The SAS data file V6.BIRTHDY (used in “Updating SAS Data Files with SYSTEM 2000 Data”) was created by using the following SAS program:

```
libname v6 'your-SAS-library';
data v6.birthdy;
  input lastname $10. firstnme $10. birthday date7.;
  format birthday date7.;
  datalines;
JONES      FRANK      22MAY53
MCVADE     CURTIS     25DEC54
SMITH      VIRGINIA   14NOV49
TURNER     BECKY     26APR50
;
```

The output shows the results after running the following program on the data file:

```
proc print data=v6.birthdy;
  title2 'SAS Data File V6.BIRTHDY';
  format birthday date7.;
run;
```

Output A3.10 SAS Data File V6.BIRTHDY

SAS Data File V6.BIRTHDY				1
OBS	LASTNAME	FIRSTNME	BIRTHDAY	
1	JONES	FRANK	22MAY53	
2	MCVADE	CURTIS	25DEC54	
3	SMITH	VIRGINA	14NOV49	
4	TURNER	BECKY	26APR50	

Data File V7.CONSULTING_BIRTHDAYS

The SAS data file V7.CONSULTING_BIRTHDAYS (used in “Updating SAS Data Files with SYSTEM 2000 Data”) was created by using the following SAS program:

```
data v7.consulting_birthdays;
  input last_name $ 1-13 first_name $ 14-26
  birthdate DATE7.;
  informat birthdate DATE7.;
  format birthdate DATE7.;
  datalines;
JOHNSON    ED            30JAN65
LEWIS      THOMAS       25MAY54
SMITH      AMANDA       02DEC60
WILSON     REBECCA      13APR58
;
```

The output shows the results after running the following program on the data file:

```
proc print data=v7.consulting_birthdays;
  title2 'V7.Consulting_Birthdays Data File';
```

```
run;
```

Output A3.11 SAS Data File V.Consulting_Birthdays

V7.Consulting_Birthdays Data File				1
obs	last_name	first_name	birthdate	
1	JOHNSON	ED	30JAN65	
2	LEWIS	THOMAS	25MAY54	
3	SMITH	AMANDA	02DEC60	
4	WILSON	REBECCA	13APR58	

Data File MYDATA.CORPHON

The SAS data file MYDATA.CORPHON (used in “Browsing and Updating SYSTEM 2000 Data”) was created by using the following SAS program:

```
libname mydata 'your-SAS-library';
data mydata.corphon;
  input lastname $15. firstnme $15. phone $10.;
  datalines;
BOWMAN      HUGH E.      109 XT901
FAULKNER    CARRIE ANN   132 XT417
GARRETT     OLAN M.      212 XT208
KNAPP       PATRICE R.   222 XT 12
KNIGHT      ALTHEA       213 XT218
MILLSAP     JOEL B.      131 XT224
MUELLER     PATSY        223 XT822
NATHANIEL   DARRYL       118 XT544
SALAZAR     YOLANDA      111 XT169
WATERHOUSE  CLIFTON P.   101 XT109
;
```

The output shows the results after running the following program on the data file:

```
proc print data=mydata.corphon;
  title 'SAS Data File MYDATA.CORPHON';
run;
```

Output A3.12 SAS Data File MYDATA.CORPHON

SAS Data File MYDATA.CORPHON				1
OBS	LASTNAME	FIRSTNME	PHONE	
1	BOWMAN	HUGH E.	109 XT901	
2	FAULKNER	CARRIE ANN	132 XT417	
3	GARRETT	OLAN M.	212 XT208	
4	KNAPP	PATRICE R.	222 XT 12	
5	KNIGHT	ALTHEA	213 XT218	
6	MILLSAP	JOEL B.	131 XT224	
7	MUELLER	PATSY	223 XT822	
8	NATHANIEL	DARRYL	118 XT544	
9	SALAZAR	YOLANDA	111 XT169	
10	WATERHOUSE	CLIFTON P.	101 XT109	

Data File Trans.Banking

The SAS data file Trans.Banking (used in 'Creating and Loading SYSTEM 2000 Databases', as input to the DBLOAD procedure to create and load data into the SYSTEM 2000 database Banking) was created by using the following SAS program:

```
libname trans 'your.SAS.library';
data trans.banking;
  input custname & $20.
        custid & $7.
        acctnum & 4.
        accttyp & $1.
        transtyp & $1.
        transamt & dollar10.2
        transdat & date7.;
  format acctnum 4.
        transamt dollar10.2
        transdat date7.;
  informat transdat date.;
  datalines;
booker, john 74-9838 8349 s d $40.00 05jun89
lopez, pat 38-7274 9896 s d $15.67 23jun89
...more data lines
;
```

The output shows the results after running the following program on the data file:

```
proc print data=trans.banking;
  title 'Data in SAS Data File TRANS.BANKING';
run;
```

Output A3.13 SAS Data File Trans.Banking

Data in SAS Data File TRANS.BANKING							
OBS	CUSTNAME	CUSTID	ACCTNUM	ACCTTYP	TRANSTYP	TRANSAMT	TRANSDAT
1	BOOKER, JOHN	74-9838	8349	S	D	\$40.00	05JUN89
2	LOPEZ, PAT	38-7274	9896	S	D	\$15.67	23JUN89
3	JONES, APRIL	85-4941	4141	C	W	\$213.78	29JUN89
4	BOOKER, JOHN	74-9838	8349	S	I	\$34.76	30JUN89
5	MILLER, NANCY	07-6163	7890	S	I	\$53.98	30JUN89
6	LOPEZ, PAT	38-7274	9896	S	I	\$16.43	30JUN89
7	JONES, APRIL	85-4941	4141	C	W	\$354.70	30JUN89
8	MILLER, NANCY	07-6163	7890	S	D	\$1,245.87	01JUL89
9	JONES, APRIL	85-4941	4141	C	D	\$2,298.65	01JUL89
10	MILLER, NANCY	07-6163	3876	C	W	\$45.98	08JUL89
11	ROGERS, MIKE	96-5052	4576	C	D	\$75.00	10JUL89
12	BOOKER, JOHN	74-9838	3673	C	D	\$150.00	10JUL89
13	LOPEZ, PAT	38-7274	9896	S	D	\$50.00	10JUL89
14	BOOKER, JOHN	74-9838	3673	C	W	\$65.43	13JUL89
15	ROGERS, MIKE	96-5052	4576	C	W	\$12.34	13JUL89
16	ROGERS, MIKE	96-5052	4576	C	W	\$45.67	13JUL89
17	MILLER, NANCY	07-6163	3876	C	D	\$56.79	14JUL89
18	ROGERS, MIKE	96-5052	4576	C	W	\$12.16	15JUL89

Note: The input SAS data file Trans.Banking must be sorted before you can use the data for the examples in [Chapter 6, “Creating and Loading SYSTEM 2000 Databases,” on page 59](#). The programs using PROC DBLOAD create and load a three-level SYSTEM 2000 database. Each logical entry represents a customer. Records at level 1 contain data for the accounts by customer; records at level 2 contain transaction data.

The following program sorts the input SAS data file Trans.Banking by the variables CUSTNAME and ACCTNUM:

```
proc sort data=trans.banking;
  by custname acctnum;
run;
```

After you sort the data file Trans.Banking, you can use it to create the database Banking (shown in [Chapter 6, “Creating and Loading SYSTEM 2000 Databases,” on page 59](#)), which also contains the new database definition and the stored data.

Recommended Reading

Here is the recommended reading list for this title:

- *Base SAS Procedures Guide*
- *SAS Language Reference: Concepts*
- *SAS SQL Procedure User's Guide*
- *SAS Companion -- for your operating system*
- *The Little SAS Book: A Primer*

For a complete list of SAS publications, go to sas.com/store/books. If you have questions about which titles you need, please contact a SAS Representative:

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-0025
Fax: 1-919-677-4444
Email: sasbook@sas.com
Web address: sas.com/store/books

Glossary

access descriptor

a SAS/ACCESS file that describes data that is managed by SAS, by a database management system, or by a PC-based software application such as Microsoft Excel, Lotus 1-2-3, or dBASE. After creating an access descriptor, you can use it as the basis for creating one or more view descriptors. *See also* [view descriptor](#).

Accounting Log

in SYSTEM 2000 software, a file used to hold the Multi-User accounting system records.

action-clause

in the SYSTEM 2000 QUEST language, the portion of a command on the left of a where-clause or the entire command if there is no where-clause; for retrievals, the retrieval-clause and the ordering-clause, if any; for updates, the update-clause.

active database

in SYSTEM 2000 software, a database that is on disk for production use in the Self-Contained Facility or in PLEX jobs.

ad hoc function

in SYSTEM 2000 QUEST software, an arithmetic expression enclosed in parentheses and specified in an action-clause.

alias

an alternative name, usually a shortened form, for a particular SAS language element. An example of an alias is PWD for PASSWORD.

allocation

the operating system's association between a logical name (DDname) and an operating system data set.

ancestor record

a record on the level that precedes a specified record in the same path.

archival database

in SYSTEM 2000 software, a copy of a database saved for use in recovery or restoration. An archival database is a historical database saved at a significant point in time.

arithmetic expression

See [SAS expression](#).

arithmetic operator

in SAS, any of the symbols (+, -, /, *, and **) that are used to perform addition, subtraction, division, multiplication, or exponentiation in arithmetic expressions. In SYSTEM 2000 software only, ** is not supported.

ASM PLEX

in the SYSTEM 2000 PLEX facility, the Programming Language Extension for IBM Assembler Language.

attach a database

in SYSTEM 2000 software, to make database files available to a job.

available space

in SYSTEM 2000 software, the allocated or extendable storage space beyond the logical end of a file or table.

batch mode

a noninteractive method of running SAS programs by which a file (containing SAS statements along with any necessary operating system commands) is submitted to the batch queue of the operating environment for execution.

by-clause

part of a retrieval-clause, which specifies that the values within a data tree be displayed together. A by-clause begins with the keyword BY and includes its focal record and all subsequent components in the retrieval-clause, up to the beginning of another by-clause or to the beginning of an ordering-clause or where-clause.

by-phrase

in SYSTEM 2000 software, a system function followed by the keyword BY and its focal record.

C-number

See [component number](#).

CHARACTER

in SYSTEM 2000 software, an item type for textual values where leading blanks, trailing blanks, and multiple blanks between words are ignored. CHARACTER is abbreviated as CHAR.

children

the records that immediately follow a specified record.

CICS Command Editor

a tool that enables you to create, save, retrieve, and submit command streams in a CICS environment. It can be used in the CICS interface to access, modify, and resubmit the most recent command stream.

CICS Interface

in SYSTEM 2000 software, an interface that allows direct communication with the IBM Customer Information Control System (CICS) TP monitor.

clearing

in SYSTEM 2000 software, the process of copying an updated page in a database or Update Log from main memory to disk.

close a database

in SYSTEM 2000 software, to make the files of a database unavailable to a job.

COBOL PLEX

in the SYSTEM 2000 PLEX facility, the COBOL Programming Language Extension.

Collect File

in SYSTEM 2000 software, a temporary file holding a relational table containing COLLECT command output. Collect File values can originate in more than one database and can be used in QUEST language and REPORT language commands.

command

a directive to an operating system to perform a particular task.

Command Editor (CICS Command Editor)

in SYSTEM 2000 software, a capability for creating, saving, retrieving, and submitting command streams in a CICS environment.

Command File

in SYSTEM 2000 software, a user file that includes Self-Contained Facility (SCF) input commands. The Command File is the standard input file by default, but it can be a user-assigned alternate file.

command terminator

in SYSTEM 2000 software, a signal that indicates the end of a command, usually a colon or semicolon.

COMMBLOCK

in a SYSTEM 2000 PLEX program, a declarative statement that sets up a COMMBLOCK. The COMMBLOCK is a data area for passing status information back and forth between the PLEX program and SYSTEM 2000 software.

component

a self-contained, reusable programming object that provides some type of service to other components in an object-oriented programming environment.

component label

the user-assigned component name or number that uniquely identifies a component in a SYSTEM 2000 database definition.

component name

a unique name that is assigned to a component in a SYSTEM 2000 database definition. *See also* [component](#).

component number (C-number)

a unique number that is assigned to a component in a SYSTEM 2000 database definition. *See also* [component](#).

condition

a part of a SYSTEM 2000 where-clause that contains an EXISTS, FAILS, EQ, NE, SPANS, LT, GT, LE, GE, or CONTAINS operator (or an equivalent symbol) and its operands, which are either schema items or specified values.

connecting string

optional syntax that you can use in a SYSTEM 2000 where-clause that is included in a SAS/ACCESS view descriptor. A connecting string tells the interface view engine how you want to connect conditions in the SYSTEM 2000 where-clause with conditions that are translated from a SAS WHERE clause.

constant

in SAS software, a number or a character string that indicates a fixed value.

CONTROL

in SYSTEM 2000 software, a Self-Contained Facility language used for administrative tasks, such as saving, restoring, and recovering databases. CONTROL is the processor of the CONTROL language.

Coordinated Recovery

in SYSTEM 2000 software, the process of reinstating a damaged database to a previous, undamaged condition. Coordinated Recovery uses the Rollback Log and the Update Log.

create a database

in SYSTEM 2000 software, to assign a name and storage space to a database yet to be defined.

cycle number (database cycle number)

in SYSTEM 2000 software, the total number of times that data have been updated or loaded into a database. This is also referred to as database cycle number.

damage flag

in SYSTEM 2000 software, an internal indicator of whether a database is damaged.

damaged database

in SYSTEM 2000 software, a database for which updates have been only partially completed before processing was stopped (for example, because of a hardware failure).

Data File

in SYSTEM 2000 software, a user file containing data for Self-Contained Facility commands that are to be executed by a SYSTEM 2000 processor. The Data File is the standard input file by default.

data length

in SYSTEM 2000 software, the number of symbols or characters in a data value.

data management software

an integrated software package that enables you to create and manipulate data in the form of databases.

data record

an identifiable set of values that are treated as a unit and which are associated with a schema record. A logical entry consists of related data records. *See also* [logical entry](#).

data set

See [SAS data set](#).

data structure

in SYSTEM 2000 software, the view of a database in terms of data items, data records, and the hierarchical relationships of the records. This is also called a logical data structure.

Data Table (DT)

in a SYSTEM 2000 database, a table containing all data values for the records in the database.

data value

a unit of character, numeric, or alphanumeric information that is stored as a single item in a data record.

data view

See [SAS data view](#).

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. See also [database definition](#).

database cycle number

See [cycle number](#).

database definition (definition)

a blueprint for the type of data that is stored in a SYSTEM 2000 database. A definition consists of schema records and related schema items, which are organized in a hierarchical structure. A definition labels the data to be stored, arranges the data into groups, and establishes relationships among the groups of data. See also [schema record](#).

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 [hierarchical structure](#).

Database Manager

in SYSTEM 2000 software, the nucleus (executive code) of the software.

DATE

in SYSTEM 2000 software, the item type for date values.

date and time format

instructions that tell SAS how to write numeric values as dates, times, and datetimes.

DBA password

a SYSTEM 2000 password that provides a level of authority between that of the master password and that of the secondary passwords. The DBA password enables the DBA to administer databases without being able to access the data that is stored in them.

DBMS

See [database management system](#).

DECIMAL

in SYSTEM 2000 software, an item type for numeric values with a fixed decimal point and an optional plus or minus sign.

DEFINE language

in SYSTEM 2000 software, a Self-Contained Facility language used to define a database.

definition

See [database definition](#).

definition number

in SYSTEM 2000 software, the total number of times that a database definition has been changed (mapped).

Definition Table

in a SYSTEM 2000 database, a table that contains the definition of a database.

delimiter

a character that serves as a boundary that separates the elements of a text string.

descendant

a record that a member that resides at a lower level in relation to other members in the hierarchy. A record is a descendant of its ancestors.

DESCRIBE order

in SYSTEM 2000 software, the order in which the schema of a database is displayed as a result of the DESCRIBE command in the QUEST language.

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.

descriptor information

information about the contents and attributes of a SAS data set. For example, the descriptor information includes the data types and lengths of the variables, as well as which engine was used to create the data. SAS creates and maintains descriptor information within every SAS data set.

Diagnostic Log

in SYSTEM 2000 software, a file that contains diagnostic tuning aids (at different levels) resulting from job processing in a Multi-User environment.

dimension level (level)

an element of a dimension hierarchy. Levels describe the dimension from the highest (most summarized) level to the lowest (most detailed) level. For example, possible levels for a Geography dimension are Country, Region, State or Province, and City.

direct mode

in SYSTEM 2000 software, the state in which the Update Log for a database is written directly to tape or disk.

disjoint record

a schema record in a SYSTEM 2000 database that is outside a path as specified in a view descriptor, and thus cannot be included.

Distinct Values Table (DVT)

in SYSTEM 2000 software, one of two tables that make up the index for a database. This table contains an entry for each distinct value of a key item. *See also* [Multiple Occurrence Table](#).

DOUBLE

in SYSTEM 2000 software, an item type for double-precision, floating-point values.

DT

See [Data Table](#).

DVT

See [Distinct Values Table](#).

DYNAMIC where-clause

in SYSTEM 2000 software, a QUEST where-clause that can be created and modified within a PLEX program while the program is executing.

EFT

See [Extended Field Table](#).

engine (SAS engine)

a component of SAS software that reads from or writes to a file. Various engines enable SAS to access different types of file formats.

entry terminator

in SYSTEM 2000 software, a signal that indicates the end of a data entry or subtree in the Data File. The default entry terminator is END*.

exclusive use

the condition under which only one user can access a database at a time.

execution option

in SYSTEM 2000 software, an option specified in a JCL EXEC statement and then passed as a parameter to the program called in the job step.

execution parameter

in SYSTEM 2000 software, an independent variable, such as an argument in a string, subroutine, or execution statement, for which a value is assigned at time of use.

Extended Field Table (EFT)

in a SYSTEM 2000 database, a table used to store the characters that extend beyond the assigned field width.

family

a SYSTEM 2000 record, all its ancestors, and all its descendants.

focal record

in SYSTEM 2000 software, a data record that determines the family of schema records in a has-expression, a by-clause, or a by-phrase.

format a database

in SYSTEM 2000 software, to fill the pages of the database tables with binary zeros.

FORTRAN PLEX

in the SYSTEM 2000 PLEX family, the FORTRAN Programming Language Extension.

full pass

in SYSTEM 2000 software, the process of examining all the records in the Data Table one by one.

function

See [SAS function](#).

Genius

in SYSTEM 2000 software, a conversational facility for specifying LISTING reports.

global hold

in SYSTEM 2000 software, the state in which an entire database is reserved for use by one user temporarily to prevent other users from updating or reserving any portion of the database.

has-clause

in a SYSTEM 2000 where-clause, the word HAS preceded by a record name or component number and followed by a condition or expression.

hierarchical data management

the practice of storing and accessing data in a database structure that minimizes redundancy by organizing stored data in levels.

hierarchical structure

an arrangement of data in which records occur at distinct levels, with different types of information at each level. Records are related to other records as ancestors, descendants, siblings, and so on.

hierarchical table (HT)

in a SYSTEM 2000 database, a table that contains the hierarchical relationships among the records in a database.

HOLD option

in SYSTEM 2000 software, an option in a PLEX retrieval command that enables a user to reserve specified data records temporarily.

HT

See [hierarchical table](#).

if-clause

in the SYSTEM 2000 QUEUE and REPORT languages, an additional criterion to further qualify the records that satisfy a where-clause.

immediate mode

in SYSTEM 2000 software, the PLEX mode in which an update command is processed immediately after being issued. See also [load mode](#), [queue mode](#).

inclusion list

in SYSTEM 2000 software, a part of a PLEX retrieval or update command specifying the items that are to participate in the operation.

incremental loading

in SYSTEM 2000 software, the loading of new logical entries in separate batches.

index

See [SAS index](#).

informat

See [SAS informat](#).

initial loading

in SYSTEM 2000 software, the loading of the first batch of logical entries into the database. See also [incremental loading](#).

INTEGER

in SYSTEM 2000 software, an item type for whole numeric values (or 0) having an optional plus or minus sign.

interactive line mode (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 type of SAS engine that SAS/ACCESS software uses to retrieve data from files that have been formatted by another vendor's software. Each SAS/ACCESS interface has its own interface view engine, which reads the interface product data and returns the data in a form that SAS can understand (that is, in a SAS data set). See also [engine](#).

invoke a string

in SYSTEM 2000 software, to initiate the transaction stored in a string by giving the string's name or number and supplying any parameter values that were defined.

Item Menu

in SYSTEM 2000 software, a QueX screen containing the items available to a specific user for a particular record and a menu of commands that may be used in processing the items displayed.

item type

a classification of values that determines how the values will be stored in a SYSTEM 2000 database. The item types are CHARACTER, TEXT, INTEGER, DECIMAL, MONEY, DATE, REAL (or FLOAT), DOUBLE, and UNDEFINED.

Keepfile

in SYSTEM 2000 software, a file that holds the permanent Update Log recordings; it is used to recover a database.

key

See [lookup key](#).

key condition

in SYSTEM 2000 software, a condition that contains a key item, which means records are qualified by means of the index.

left sibling

in a SYSTEM 2000 schema or data tree, the sibling record on the left of a given record, that is, the record one position nearer to the beginning of the logical chain of siblings.

level

See [dimension level](#).

library member

any of several types of SAS file in a SAS library. A library member can be a data set, a view, a catalog, a stored program, or an access descriptor.

library reference

See [libref](#).

libref (library reference)

a SAS name that is associated with the location of a SAS library. For example, in the name MYLIB.MYFILE, MYLIB is the libref, and MYFILE is a file in the SAS library. See also [SAS library](#).

line mode

See [interactive line mode](#).

load a database (populate a database)

in SYSTEM 2000 software, to enter logical entries in a database.

load mode

in SYSTEM 2000 software, the PLEX mode in which only INSERT commands are issued. The software processes those commands in part when they are issued and completes processing when the TERMINATE command is given. See also [immediate mode](#), [queue mode](#).

loader stream

in SYSTEM 2000 software, the values for logical entries coded in the format necessary to load a database.

local hold

in SYSTEM 2000 software, the state in which one data record is reserved temporarily for use by a user to prevent other users from updating or reserving that data record.

Locate File

in SYSTEM 2000 software, a PLEX work file that stores addresses of data records for subset processing.

logical entry

the data records that pertain to one entry in a SYSTEM 2000 database. For example, in the EMPLOYEE database, all data records that pertain to one employee comprise a logical entry.

logical order

in SYSTEM 2000 software, the order of records in a tree in which each record precedes all of its descendants and all of its right siblings.

logical unit of work

in SYSTEM 2000 software, a sequence of updates bounded by synchpoints. A logical unit of work is used in Coordinated Recovery.

lookup key (key)

a value that uniquely identifies a specific record and its order among other records in a database or table.

master password

the password under which a SYSTEM 2000 database is created. The holder of the master password can access the entire database and has the authority to use any SYSTEM 2000 statement.

Master Record

in SYSTEM 2000 software, a table of identification information, such as the database name and passwords.

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.

Message File

in SYSTEM 2000 software, a user file containing messages issued by the software, such as error diagnostics, informative messages, and echoes of commands. The Message File is the standard output file by default, but it can be a user-assigned alternate file.

missing value

a type of value for a variable that contains no data for a particular row or column. By default, SAS writes a missing numeric value as a single period and a missing character value as a blank space. *See also* [null value](#), [null item](#), [null record](#).

MONEY

in SYSTEM 2000 software, an item type for money values; it is similar to DECIMAL type, except that \$, CR, and DB are displayed with the values.

MOT

See [Multiple Occurrence Table](#).

multi-user environment

a data entry 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](#).

Multiple Occurrence Table (MOT)

in SYSTEM 2000 software, a table that is part of the index.

non-exclusive use

in SYSTEM 2000 software, the state in which more than one user can perform updates and retrievals on the same database.

non-key condition

in SYSTEM 2000 software, a where-clause condition for which the software uses the Data Table instead of the index to qualify records.

normalization

the process of obtaining ancestors or descendants of qualified records during the processing of a SYSTEM 2000 where-clause.

NULL

in SYSTEM 2000 software, a format option that requests the output display of NULL for items having missing values. *See also* [null value](#).

null item

an item for which space is allocated in a record, although no value currently exists in the SYSTEM 2000 database. A null item is similar to a SAS missing value, but they are not identical. *See also* [missing value](#), [schema item](#).

null record

a data record that contains all null items.

null value

a special value that indicates the absence of information. Null values are analogous to SAS missing values. *See also* [missing value](#).

numeric item

in SYSTEM 2000 software, an item whose values are numbers that can be calculated (as opposed to ZIP codes). The numeric item types are INTEGER, DECIMAL, MONEY, REAL (FLOAT), and DOUBLE.

observation

a row in a SAS data set. All of the data values in an observation are associated with a single entity such as a customer or a state. Each observation contains either one data value or a missing-value indicator for each variable.

open a database

in SYSTEM 2000 software, to attach a database for use.

ORDERED BY clause

See [ordering-clause](#).

ordering-clause (ORDERED BY clause)

a set of one or more user-specified SYSTEM 2000 schema items that control the sorting of selected values.

ordinal

in SYSTEM 2000 software, the place of a member in an ordered set, for example, the left-to-right ordering of data records within a set of similar siblings, the left-to-right ordering of conditions in a PLEX where-clause, or the first-to-last ordering of items in a subschema record.

padding

in SYSTEM 2000 software, the unused space in a page of a database table that is reserved for future use.

parametric string

in SYSTEM 2000 software, a string containing one or more parameters for which values are assigned when the string is invoked.

path

See [view descriptor path](#).

physical order

the order in which data records or observations appear in their storage structure.

physical storage structure

in SYSTEM 2000 software See also [storage structure](#).

picture (schema item picture)

the logical size (length) of values for a particular schema item in a SYSTEM 2000 database.

PL/I PLEX

in SYSTEM 2000 software, an abbreviation for PL/I Programming Language Extension, a dialect of the PLEX Facility.

PLEX

in SYSTEM 2000 software, a facility for extending a COBOL, FORTRAN, PL/I, or Assembler program to include SYSTEM 2000 commands. PLEX is also the name of the processor of the PLEX language.

PLEX processor directive

in SYSTEM 2000 software, an instruction issued to the PLEX processor that does not result in executable code.

populate a database

See [load a database](#).

Primary Record

in SYSTEM 2000 QueX software, the Request Record used with a SELECT command to initiate a sequence of retrievals. All other records requested must be related to the Primary Record in some manner.

processor

in SYSTEM 2000 software, a subsystem that is specific to each of the languages available, for example, DEFINE, CONTROL, QUEST, REPORT, and PLEX.

Program Service Processor (PSP)

in SYSTEM 2000 software, the interface subsystem between an executing PLEX program and the Data Base Manager.

PSP

See [Program Service Processor](#).

qualified record

in SYSTEM 2000 software, a record that satisfies one or more conditions in a where-clause.

QUEST

the query/update language that is used in SYSTEM 2000 software. QUEST is also the name of the processor of the QUEST language.

QUEUE

in SYSTEM 2000 software, a QUEST language option used for queries and updates in which all commands for a session are processed simultaneously when the TERMINATE command is given. QUEUE is also the name of the processor for the QUEUE language.

queue mode

in SYSTEM 2000 software, a PLEX mode in which update commands are not processed until the TERMINATE command is given. *See also* [immediate mode](#), [load mode](#).

QueX database

in SYSTEM 2000 software, a database that contains a description of each QueX user view of the data. A QueX database is used to condition and control the QueX environment.

QueX menu

in SYSTEM 2000 software, a QueX screen containing the records available to a specific user.

QueX software

in SYSTEM 2000 software, a query and update facility that is fixed-screen, interactive, and menu-driven.

QueX table

in SYSTEM 2000 software, any of the run-time tables that drive QueX software.

R-authority

a code that is specified by the holder of the master password and which gives the holder of a secondary password the authority to retrieve a SYSTEM 2000 schema component.

REAL

in SYSTEM 2000 software, an item type for single-precision, floating-point values.

record membership

in SYSTEM 2000 software, the relationship between an item and the record that it belongs to.

record relationship

in SYSTEM 2000 software, the hierarchical relationship between records.

recover a database

in SYSTEM 2000 software, to restore an archival database and apply some or all of the updates recorded in its Keepfile.

reinstate a database

in SYSTEM 2000 software, to use the Coordinated Recovery process to bring a damaged database back to a previous, undamaged state.

related

in SYSTEM 2000 software, the state of belonging to the same schema path.

release a database

in SYSTEM 2000 software, to free the space occupied by the database files on disk.

reload a database

in SYSTEM 2000 software, to unload a database, then build new database tables with the unloaded data by means of the RELOAD command.

rename a database

in SYSTEM 2000 software, to change the name of a database and its files.

reorganize a database

in SYSTEM 2000 software, to compress the index of a database.

REPORT

in SYSTEM 2000 software, a language for defining and generating reports. REPORT is also the name of the processor of the REPORT language.

Report File

in SYSTEM 2000 software, a user file containing the results of retrieval requests, such as reports, tallies, and the output from the REPORT processor. The Report File is the standard output file by default, but it can be a user-assigned alternate file.

Request Record

in SYSTEM 2000 QueX software, the record requested for display.

restore a database

in SYSTEM 2000 software, to copy an archival database disk for active use.

restructure a database

in SYSTEM 2000 software, to change the logical data structure and the physical storage structure of a database to reflect a schema modification.

retrieval-clause

in the SYSTEM 2000 QUEST language, the list of components to be retrieved.

return code

a numeric value that indicates whether a request was successful. A return code can also indicate a specific error or warning.

reusable space

in SYSTEM 2000 software, the inactive storage space within a database table made available by deletions of values or records.

right sibling

in a SYSTEM 2000 schema or data tree, the sibling record on the right of a given record, that is, the record one position farther from the beginning of the logical chain of siblings.

rollback

a data recovery process that restores a database after a hardware or software failure, or that returns it to a state before changes were made.

rollback log

in SYSTEM 2000 software, a file that holds copies of database pages as they appeared before the update commands were applied.

root

See [root node](#).

root node (root)

the topmost level in a hierarchical tree, representing the entire tree and its contents.

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 (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. See also [descriptor information](#).

SAS data view (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 engine

See [engine](#).

SAS expression (arithmetic expression)

a type of macro expression consisting of a sequence of operands and arithmetic operators that form a set of instructions that are evaluated to produce a numeric value, a character value, or a Boolean value. Examples of operands are constants and system functions. SAS uses arithmetic expressions in program statements to create variables, to assign values, to calculate new values, to transform variables, and to perform conditional processing.

SAS file

a specially structured file that is created, organized, and maintained by SAS. A SAS file can be a SAS data set, a catalog, a stored program, an access descriptor, a utility file, a multidimensional database file, a financial database file, a data mining database file, or an item store file.

SAS function (function)

a type of SAS language element that is used to process one or more arguments and then to return a result that can be used in either an assignment statement or an expression.

SAS index (index)

a component of a SAS data set that enables SAS to access observations in the SAS data set quickly and efficiently. The purpose of SAS indexes is to optimize WHERE-clause processing and to facilitate BY-group processing.

SAS informat (informat)

a type of SAS language element that is used to read data values according to the data's type: numeric, character, date, time, or timestamp.

SAS library

one or more files that are defined, recognized, and accessible by SAS, and that are referenced and stored as a unit. Each file is a member of the library.

SAS variable (variable)

a column in a SAS data set or in a SAS data view. The data values for each variable describe a single characteristic for all observations (rows).

save a database

in SYSTEM 2000 software, to copy a database from disk to a permanent file called the Savefile.

Savefile

in SYSTEM 2000 software, a tape or disk file that holds the archival database. After you save a database, the Savefile can be used to restore the database at a later time.

SCF

See [Self-Contained Facility](#).

schema

a map or model of the overall data structure of a database. A schema consists of schema records that are organized in a hierarchical tree structure. Schema records contain schema items.

schema component

in SYSTEM 2000 software, a schema item or a schema record.

schema item

a component that specifies the name and characteristics of a group of SYSTEM 2000 database values. That is, a schema item has a name, a type, and a picture (length). Each value stored in a SYSTEM 2000 database corresponds to a schema item. A SYSTEM 2000 schema item is analogous to a SAS variable.

schema item picture

See [picture](#).

schema path

in SYSTEM 2000 software, a path in a schema tree.

schema record (SR)

an identifiable set of associated schema items that are treated as a unit in a SYSTEM 2000 database. See also [disjoint record](#).

secondary password

a password, other than the master password or DBA password, that restricts SYSTEM 2000 statement usage and which specifically assigns update, retrieval, and where-clause authorities for any or all components of a SYSTEM 2000 database.

Secondary Record

in SYSTEM 2000 QueX software, a Request Record that is related to the Primary Record as an ancestor, as a descendant, or as part of a linked network.

Security by Entry

in SYSTEM 2000 software, a security feature that protects the entire database. To access any given logical entry, you must know the private value for a certain item.

Self-Contained Facility (SCF)

in SYSTEM 2000 software, the facility consisting of the DEFINE, CONTROL, QUEST, QUEUE, and REPORT languages. The SCF languages do not interact with formal programming languages. *See also* [PLEX](#).

separator

in SYSTEM 2000 software, a special character that separates labels from values in commands, value streams, and output.

session

a single period during which a software application is in use, from the time the application is invoked until its execution is terminated.

sibling

in a hierarchical database, any of two or more segments or records that have the same parent segment or record.

similar records

in SYSTEM 2000 software, data records that are occurrences of the same schema record.

single-user environment

a SYSTEM 2000 execution environment in which you are working with your own copy of SYSTEM 2000 software. In a single-user environment, you usually have exclusive access to the database. However, the single-user environment can be configured so that multiple users can query the database. *See also* [multi-user environment](#).

source record

in the SYSTEM 2000 PLEX facility, a subschema record that appears on the left side of a LINK command.

SR

See [schema record](#).

SSR

See [subschema record](#).

stack

in the SYSTEM 2000 PLEX facility, an area in which the software saves the addresses of the most recently retrieved data records. The effect of every PLEX retrieval and update command depends on the contents of the stack.

storage structure

in SYSTEM 2000 software, the stored version of a database, the physical files, indexes, pages, pointers, and fields. This is also called physical storage structure.

stored function

in SYSTEM 2000 software

stored string

a text string that is contained in a SYSTEM 2000 database definition and which can be invoked by using the string number or name.

string

a stored command or part of a command or series of commands that are invoked by specifying the string name or number in another command. Strings are part of a SYSTEM 2000 database definition

subschemata

in a SYSTEM 2000 PLEX program, one specific application's view of the database. A subschemata consists of a collection of subschemata records.

subschemata item

in SYSTEM 2000 software, an item in a subschemata record.

subschemata record (SSR)

in a SYSTEM 2000 PLEX program, the application's view of a schemata record. A subschemata record contains subschemata items; each item corresponds to a schemata item in the database definition.

subtree

in SYSTEM 2000 software, a tree consisting of a given record and all its descendants.

synchronpoint

in SYSTEM 2000 software, a point at which the software marks the end of a series of updates. Up to the most recent synchronpoint, the database is secure and the updates are available for Coordinated Recovery.

syntax

a set of rules specifying proper construction of statements or commands.

SYSTEM 2000 view

a file that reads data directly from a SYSTEM 2000 database.

system function

in SYSTEM 2000 software, an operation available for use in an arithmetic expression or action-clause. The six system functions are AVG, COUNT, MAX, MIN, SIGMA, and SUM. A system function can operate on an arithmetic expression or a schemata item.

system release number

in SYSTEM 2000 software, a number indicating the version of the software used to create the database. The DESCRIBE command prints the system release number.

system separator

in SYSTEM 2000 software *See also* [separator](#).

system string

in SYSTEM 2000 software, syntax that provides the current date and time according to the computer's internal calendar and clock. The four system strings are *NOW*, *TODAY*, *FTODAY*, and *DATA*.

system-wide command

in SYSTEM 2000 software, a command accepted by any processor in the Self-Contained Facility.

target level

in SYSTEM 2000 software, the level of the data records to be selected for the action-clause.

target record

in SYSTEM 2000 software, the schema record that determines the type of records to be selected for an action-clause. The target record determines the target level. In the PLEX facility, the subschema record on the right side of the LINK verb.

TEXT

in SYSTEM 2000 software, an item type for textual values in which blanks are retained. You must include the blanks when specifying the value in a where-clause. For example, JOHN SMITH is not the same value as JOHN SMITH.

text search

in SYSTEM 2000 software, a capability that permits the search for data patterns within subsets of values.

tree

a hierarchical file structure that has a branching structure reminiscent of a physical tree.

U-authority

a code that is set by the holder of the master password and which gives the holder of a secondary password the authority to update a SYSTEM 2000 schema item or schema record.

UNDEFINED item type

in SYSTEM 2000 software, an item type for hexadecimal values. All EBCDIC characters from x'00' through x'FF' are valid.

unload a database

in SYSTEM 2000 software, to write data from a database to the Report File in loader stream format.

Update Log

in SYSTEM 2000 software, the software's journal of update processing.

user file

in SYSTEM 2000 software, any of these files: Command File, Data File, Message File, or Report File.

user view

in SYSTEM 2000 QueX software, the data and environment for a specific user. The user view is the subset of items, record, record relationships, and options defined from that user's perspective.

value stream

in SYSTEM 2000 software, values coded in a format the software can read in order to update a database.

variable

See [SAS variable](#).

via-clause

in the SYSTEM 2000 PLEX facility, the part of a LINK command that states the conditions for establishing a link between two subschema records.

view

a definition of a virtual data set that is named and stored for later use. A view contains no data; it merely describes or defines data that is stored elsewhere.

view descriptor

a SAS/ACCESS file that defines part or all of the DBMS data that is described by an access descriptor. *See also* [access descriptor](#).

view descriptor path (path)

a reference in code to a particular record and all of its hierarchical ancestors in a SYSTEM 2000 database. *See also* [disjoint record](#).

W-authority

a code that is set by the holder of the master password and which gives the holder of a secondary password the authority to use SYSTEM 2000 schema items or schema records for selection criteria in a where-clause.

where-clause

a set of one or more conditions that users specify as selection criteria for SYSTEM 2000 updates or retrievals.

Index

-
- A**
- ACCDESC= option
 - PROC ACCESS statement 20, 71
 - ACCDESC= statement 96
 - DBLOAD procedure 96
 - ACCESS 70
 - access descriptors 2, 19
 - creating 70, 80
 - creating view descriptors from 70, 71
 - dropping variables from 81
 - effects of changing database definitions 116
 - example data 139
 - identifying 71
 - libref and member name 63
 - listing items and item information 84
 - naming 96
 - passwords for 68, 71
 - selecting items for view descriptors 87
 - access mode 109
 - See also* [single-user mode](#)
 - overriding 122
 - specifying 102
 - storing 89
 - ACCESS procedure 1, 68
 - ASSIGN= statement 79
 - BYKEY statement 80
 - CREATE statement 70, 80
 - creating descriptor files 20
 - data conversions 78
 - database specification 81
 - DATABASE statement 70, 81
 - database-description statements 68, 70
 - DROP statement 81
 - editing statements 68, 70
 - efficient use of view descriptors 78
 - FORMAT statement 82
 - INFORMAT statement 83
 - interface view engine and 115
 - LENGTH statement 84
 - LIST statement 84
 - ordering-clause 77
 - passwords for descriptor files 68
 - PROC ACCESS statement 71
 - QUIT statement 85
 - RENAME statement 86
 - RESET statement 86
 - S2KPW statement 89
 - SELECT statement 87
 - statements 70
 - SUBSET statement 88
 - terminating 85
 - UNIQUE statement 90
 - where-clause 72
 - aggregate functions 34
 - ALLOC command 93
 - alphanumeric values
 - blanks not removed 14
 - blanks removed 13
 - ambiguous inserts 47
 - ancestor records 9
 - AND operator
 - where-clause 75
 - APPEND procedure 52
 - appending data to data files 52
 - appending data to view descriptors 56
 - Version 7 (or later) files 55
 - WHERE statement 54
 - appending data 52
 - to data files 52
 - to view descriptors 56
 - Version 7 (or later) files 55
 - ASSIGN_statement 79
 - ASSIGN= statement
 - ACCESS procedure 79
 - AT operator
 - connecting strings 135
 - where-clause 76
 - attention interrupts, TSO
 - Multi-User mode 110
 - single-user mode 108
 - authorities 16
 - browsing and updating data 41
- B**
- BASE= data set
 - appending data and 52

- batch mode
 - DBLOAD procedure 92
 - binary bit-string data 14
 - binary operators
 - where-clause 75
 - BLANKS option
 - PROC QUEST statement 110
 - browsing data 41
 - authorities 41
 - directly from SAS 41
 - FSBROWSE procedure 42
 - FSVIEW procedure 42, 44
 - passwords 41
 - QUEST procedure 56
 - SAS/FSP procedures 42
 - SQL procedure 49
 - subsetting data 45
 - view descriptors for 44
 - WHERE clause and 45
 - BY clause 15
 - BY key 65, 123
 - examples 124
 - performance and 125
 - BY statement 77
 - BYKEY 47
 - BYKEY statement
 - ACCESS procedure 80
 - BYKEY_statement 79
- C**
- C-numbers 8
 - calendar dates 13
 - CHARACTER item type 13
 - character item types 13
 - character width 84
 - CHART procedure 26
 - charting data 26
 - children records 10
 - CNTLLEV= data set option 119
 - column names
 - defaulting to labels 98
 - length of 101
 - resetting to default 101
 - columns
 - renaming 100
 - combining data
 - from various sources 31
 - Command File 107, 109
 - compatibility 35, 59
 - component names and numbers 8
 - concatenating data sets 52
 - connecting strings 133
 - AT operator 135
 - HAS operator 135
 - NON-KEY specification 135
 - NOT operator 132
 - omitting a WHERE clause 135
 - OR operator 135
 - syntax 134
 - CONTAINS operator
 - where-clause 75
 - CONTENTS procedure 24
 - control nodes 12
 - conversions
 - ACCESS procedure data 78
 - CREATE INDEX statement 14
 - CREATE statement 97
 - ACCESS procedure 70, 80
 - DBLOAD procedure 97
 - CREATE_statement 80
 - customized view descriptors 92
- D**
- data
 - See also* subsetting data
 - appending 52
 - binary bit-string 14
 - browsing 41
 - charting 26
 - extracting 38, 78
 - null data 12
 - printing 25
 - redundancy 61, 123
 - sorting 39
 - transaction data 35
 - data conversions 78
 - data files
 - appending data to 52
 - loading input data files 60
 - updating 35
 - data records 10
 - deleting 122
 - inserting 123
 - inserting and deleting 47
 - null records 12
 - data security 15, 117
 - SAS security 118
 - SYSTEM 2000 security 118
 - data set options
 - CNTLLEV= 119
 - overriding corresponding values 121
 - RENAME= 52
 - S2KMODE= 122
 - S2KPW= 122
 - WHERE= 54
 - data sets
 - appending data and 52
 - concatenating 52
 - creating and loading databases from 59
 - Data Table 16

- data types
 - formats based on 79
 - resetting to default 101
 - variable names based on 79
- DATA= data set
 - appending data and 52
- DATA= option
 - PROC DBLOAD statement 94
- database administration
 - changing database definitions 116
 - changing passwords 116
 - data security 117
 - enabling rollback log 118
 - locking record levels 119
 - performance 120
 - SAS/ACCESS interface 113
- database definitions 6
 - changing, effects on descriptors 116
 - creating 97
 - creating and loading databases 59
 - database names 7
 - DBLOAD procedure and 92
 - example data 137
 - labeling data 7
 - showing 60
- database files 16
 - allocating 93
- database names 7, 97
- DATABASE statement
 - ACCESS procedure 70, 81
- DATABASE_statement 81
- databases 6
 - See also loading databases*
 - accessing without view descriptor 56
 - column names defaulting to labels 98
 - creating 60, 91
 - creating and loading from data sets 59
 - database files 16
 - DBLOAD procedure 59
 - deleting data records 47, 122
 - descriptor files 19
 - grouping schema items 8
 - grouping schema records 8
 - inserting data records 47, 123
 - KEY/NON-KEY status 98
 - labeling data 7
 - length of column names 101
 - level number for variables 99
 - loading input data files 60
 - logical entries 10
 - logical entries, adding 65, 66
 - logical entries, updating 65
 - mapping data between SAS and SYSTEM 2000 11
 - naming 7, 97
 - null data 12
 - passwords 103
 - relationships with descriptors 2
 - renaming columns 100
 - renaming variables 59
 - sorting output 15
 - specifying for ACCESS procedure 81
 - subsetting data 14, 62, 104
- DATASETS procedure
 - getting information about variables 24
 - MODIFY statement 69
 - modifying passwords 69
- DATE item type 13
- date values 13
- DBA passwords 16
- DBLOAD 94
- DBLOAD procedure 1, 59, 95
 - ACCDESC= statement 96
 - allocating database files 93
 - batch mode 92
 - CREATE statement 97
 - creating databases 60
 - customized view descriptors 92
 - database definitions and 92
 - DBN= statement 97
 - DELETE statement 97
 - disjoint schema records 94
 - INDEX statement 64, 98
 - interactive-line mode 92
 - interface view engine and 115
 - item types 92
 - LABEL statement 98
 - LEVEL statement 64, 99
 - LIST statement 99
 - LOAD statement 100
 - loading databases 63
 - loading input data files 60
 - loading one database from another 94
 - passwords 63
 - PROC DBLOAD statement 63, 94
 - processing mode for loading data 66
 - QUIT statement 100
 - RENAME statement 59, 64, 100
 - required statements 94
 - RESET statement 101
 - S2KLEN statement 101
 - S2KLOAD statement 65, 66, 102
 - S2KMODE= statement 102
 - S2KPW= statement 103
 - statements 94
 - terminating 100
 - Version 6 compatibility 59
 - VIEWDESC= statement 103
 - WHERE statement 104
- DBMS= option
 - PROC ACCESS statement 71
 - PROC DBLOAD statement 94

- DBN statement 97
 - DBN= statement
 - DBLOAD procedure 97
 - DECIMAL item type 13
 - DELETE command 47, 122
 - DELETE statement 97
 - DBLOAD procedure 97
 - SQL procedure 49, 51, 52
 - deleting data records 47, 122
 - descendant records 9
 - deleting data and 47, 49
 - DESCRIBE statement
 - QUEST procedure 60
 - descriptor files
 - access descriptors 19
 - creating 20
 - defining 19
 - overview 19
 - passwords for 68
 - subsetting data in 19
 - view descriptors 19
 - descriptors 2
 - disjoint schema records 10, 94
 - Distinct Values Table 16
 - DOUBLE item type 13
 - double-precision numbers 13
 - double-word numbers 13
 - DROP statement
 - ACCESS procedure 81
 - DROP_statement 81
- E**
- EBCDIC characters 14
 - echo of SYSTEM 2000 statements 107
 - ECHO ON/ECHO OFF statements
 - QUEST procedure 107
 - ENABLE ROLLBACK statement 118
 - END statement
 - DBLOAD procedure 100
 - engines
 - interface view engine 1
 - Version 6 35
 - example data 2, 137
 - execution environments 16
 - EXIT statement
 - ACCESS procedure 85
 - DBLOAD procedure 100
 - Extended Field Table 16
 - extracting data 38, 78
- F**
- family of records 10
 - filtering, successive 45
 - FIRSTOBS= option
 - PRINT procedure 26
 - floating-point numbers 13
 - FORCE= option
 - concatenating data sets 52
 - FORMAT statement
 - ACCESS procedure 82
 - FORMAT_statement 82
 - formats 78, 82
 - based on item names and data types 79
 - conversions to SYSTEM 2000 item types 92
 - FREQ procedure 27
 - FSBROWSE procedure
 - browsing data with 42
 - WHERE statement 45
 - FSEDIT procedure
 - updating data with 42, 43
 - WHERE statement 45
 - FSVIEW procedure
 - browsing and updating data with 42, 44
 - WHERE statement 45
 - fullword numbers 13
 - functions and strings, SYSTEM 2000 107
- G**
- GROUP BY clause
 - SQL procedure 34
 - grouping
 - schema items 8
 - schema records 8
- H**
- HAS operator
 - connecting strings 135
 - where-clause 76
 - Hierarchical Table 16
- I**
- IF statement, subsetting
 - WHERE clause versus 45
 - incremental loads 59, 65
 - adding or updating logical entries 65
 - INDEX statement 98
 - DBLOAD procedure 64, 98
 - indexed items 64, 98
 - indexing 14, 78
 - INFORMAT statement
 - ACCESS procedure 83
 - INFORMAT_statement 83
 - informats 78, 83
 - input data files
 - loading 60
 - subsetting data 62, 104

insert mode 66, 123
 Insert mode 123
 INSERT statement
 SQL procedure 49, 51
 inserting data records 47, 123
 INTEGER item type 13
 interactive-line mode
 DBLOAD procedure 92
 interface 1, 113
 interface view engine 1, 114
 ACCESS procedure and 115
 DBLOAD procedure and 115
 other SAS procedures and 116
 QUEST procedure and 115
 item names
 formats based on 79
 variable names based on 79
 item types 12
 character 13
 date 13
 default conversions 92
 numeric 13
 item width 84

J
 JCL statements 63

K
 key items 14, 64
 KEY/NON-KEY status 98

L
 LABEL statement 98
 DBLOAD procedure 98
 labels
 column names defaulting to 98
 labeling data 7
 LENGTH statement
 ACCESS procedure 84
 LENGTH_statement 83
 LEVEL statement 99
 DBLOAD procedure 64, 99
 levels
 changing 64
 grouping schema records into 8
 level number for variables 99
 LIST statement 84, 99
 ACCESS procedure 84
 DBLOAD procedure 99
 QUEST procedure 61
 LOAD statement 100
 DBLOAD procedure 100
 loading databases 63, 91

deleting specified variables 97
 executing the load 100
 from data sets 59
 from different paths 94
 incremental loads 59, 65
 loading one database from another 94
 optimized load mode 66, 102
 processing mode for 66
 subsetting input data 104
 loading input data files 60
 locking record levels 119
 log
 echo of SYSTEM 2000 statements 107
 rollback log 16, 118
 update log 16
 logical entries 10, 92
 adding data records to 66
 adding to database 65, 66
 null data in 12
 subsetting data 14
 updating 65

M
 mapping data
 between SAS and SYSTEM 2000 11
 master passwords 15
 Master Record and Definition Table 16
 MCS (multiple command submission)
 111
 MCS [procedure statement 111
 MCS statement
 QUEST procedure 111
 MEANS procedure 28
 missing values 12, 47, 126
 in selection criteria 127
 retrieving 126
 updating 126
 MODIFY statement
 DATASETS procedure 69
 MONEY item type 13
 Multi-User environment 16, 64
 Multi-User mode 102, 109
 Command File 109
 temporary output file 109
 TSO attention interrupts 110
 multiple command submission (MCS)
 111
 Multiple Occurrence Table 16

N
 NON-KEY specification 135
 NON-KEY/KEY status 98
 NOT operator
 connecting strings 132

- WHERE clause 132
 - where-clause 76
 - null data 12
 - null items 12
 - null records 12
 - nulls 12, 126
 - from deleting records 47
 - in selection criteria 127
 - retrieving 126
 - updating 126
 - numeric item types 13
- O**
- OBS= option
 - PRINT procedure 25
 - observations
 - terminology 23
 - optimized load mode 66, 102
 - inserting records 123
 - loading one database from another 94
 - options
 - overriding 121
 - OR operator
 - connecting strings 135
 - where-clause 75
 - order of processing
 - where-clause 76
 - ordering data 77
 - ordering-clause 15, 77
 - efficient use of 78
 - example 77
 - syntax 77
 - OUT= option
 - PROC ACCESS statement 71
 - output
 - showing database definitions 60
 - sorting 15
 - overriding
 - access modes 122
 - options 121
 - passwords 122
- P**
- parent records 9
 - passwords 15
 - access descriptors 68, 71
 - assigning 69, 103
 - browsing and updating data 41
 - changing 69, 116
 - clearing 69
 - DBA passwords 16
 - DBLOAD procedure 63
 - descriptor files 68
 - master passwords 15
 - modifying 69
 - overriding 122
 - secondary passwords 16
 - storing 89
 - path of a record 10
 - performance
 - BY key and 125
 - maximizing 120
 - view descriptors and 38, 78
 - picture 12
 - PLEX commands 114
 - post-processing 46
 - PRINT procedure 25
 - FIRSTOBS= option 26
 - OBS= option 25
 - printing data 25
 - printing a sample 25
 - PROC ACCESS statement 71
 - ACCDESC= option 20, 71
 - DBMS= option 71
 - OUT= option 71
 - VIEWDESC= option 72
 - PROC DBLOAD statement 63, 94
 - DATA= option 94
 - DBMS= option 94
 - PROC QUEST statement 105
 - BLANKS option 110
 - S2KMODE= option 110
 - processing modes 66
 - processing order
 - where-clause 76
- Q**
- QUEST interactive language 56
 - QUEST procedure 1, 105
 - browsing and updating data 56
 - Command File 107
 - DESCRIBE statement 60
 - ECHO ON/ECHO OFF statements 107
 - interface view engine and 115
 - LIST statement 61
 - MCS statement 111
 - Multi-User mode 109
 - PROC QUEST statement 105
 - QUIT statement 112
 - SCS statement 112
 - showing database definitions 60
 - single-user mode 108
 - statements 106
 - strings and functions 107
 - syntax 110
 - SYSTEM 2000 statements 106
 - QUEST_procedure 110
 - QUIT procedure 111
 - QUIT statement 85, 100

- ACCESS procedure 85
- DBLOAD procedure 100
- QUEST procedure 112

- R**
- RANK procedure 29
- REAL item type 13
- record-level locking 119
- records 10
 - See also [schema records](#)
 - ancestor records 9
 - children records 10
 - descendant records 9
 - family of 10
 - null records 12
 - parent records 9
 - path of 10
- redundancy 61, 123
- REMOVE INDEX statement 14
- RENAME statement 85, 100
 - ACCESS procedure 86
 - DBLOAD procedure 59, 64, 100
- RENAME= data set option
 - appending data and 52
- RENAME= option
 - UPDATE statement 37
- renaming
 - columns 100
 - variables 59, 64, 86
- RESET statement 86, 101
 - ACCESS procedure 86
 - DBLOAD procedure 101
- rollback log 16, 118

- S**
- S2KLEN statement 101
 - DBLOAD procedure 101
- S2KLOAD statement 102
 - DBLOAD procedure 65, 66, 102
- S2KMD= data set option 122
- S2KMODE statement 102
- S2KMODE= data set option 122
- S2KMODE= option
 - PROC QUEST statement 110
- S2KMODE= statement
 - DBLOAD procedure 102
- S2KPARMS file 108
- S2KPW statement 89
 - ACCESS procedure 89
- S2KPW= data set option 122
- S2KPW= statement 103
 - DBLOAD procedure 103
- SAS
 - browsing and updating data from 41
 - mapping data between SYSTEM 2000 and 11
 - security 118
- SAS data views 19
- SAS/ACCESS interface 113
- SAS/FSP procedures
 - browsing and updating data 42
 - inserting and deleting data records 47
 - WHERE statement 45
- schema components 8
- schema items 7
 - grouping 8
 - indexing 14
 - item types 12
 - key items 14
 - null items 12
 - variable names created from schema
 - item names 8
- schema records 8
 - disjoint records 10, 94
 - family of 10
 - grouping into levels 8
 - grouping schema items into 8
 - no schema items in 12
 - path of 10
 - relationships among 8
 - types of 9
- SCS (single command submission) 112
- SCS statement 112
 - QUEST procedure 112
- secondary passwords 16
- security
 - data security 15, 117
 - SAS security 118
 - SYSTEM 2000 118
- SELECT statement 87
 - ACCESS procedure 87
 - SQL procedure 49
- selection criteria 14, 78
 - nulls in 127
 - WHERE clause versus where-clause 133
- SEPARATOR IS statement 108
- single command submission (SCS) 112
- single-precision numbers 13
- single-statement queuing mode 112
- single-user environment 16
- single-user mode 102, 108
 - S2KPARMS file 108
 - TSO attention interrupts 108
- sort keys 15, 80
- SORT procedure 78
- sorting
 - data 39
 - output 15
- source data set

- appending data and 52
- SQL procedure
 - browsing and updating data 49
 - combining data from various sources 31
 - creating new items 34
 - DELETE statement 49, 51, 52
 - GROUP BY clause 34
 - INSERT statement 49, 51
 - SELECT statement 49
 - subsetting data 50
 - UPDATE statement 49, 50
 - view descriptors 49
 - WHERE clause 30, 50, 52
- statement-queuing mode 111
- statistics 27
- stored strings 136
- strings and functions, SYSTEM 2000 107
- strings, connecting
 - See [connecting strings](#)
- SUBSET statement 88
 - ACCESS procedure 88
- subsetting data 14
 - for view descriptors 88
 - in descriptor files 19
 - input data 62, 104
 - SAS/FSP procedures 45
 - SQL procedure 50
- subsetting IF statement
 - WHERE clause versus 45
- successive filtering 45
- summary functions 34
- SYSTEM 2000
 - access mode 102
 - database files 16
 - databases 6
 - default item types 92
 - execution environments 16
 - indexing 14
 - interface 1
 - item types 12
 - mapping data between SAS and 11
 - overview 5
 - passwords 15
 - security 118
 - strings and functions 107
- SYSTEM 2000 statements
 - echo in log 107
 - QUEST procedure 106
 - strings and functions 107

T

- target data set
 - appending data and 52
- terminology 23

- ternary operators
 - where-clause 75
- TEXT item type 14
- time values 13
- transaction data 35
- truncated variable names 59
- TSO attention interrupts
 - Multi-User mode 110
 - single-user mode 108

U

- unary operators
 - where-clause 74
- UNDEFINED item type 14
- UNIQUE statement 89
 - ACCESS procedure 90
- update log 16
- UPDATE statement
 - RENAME= option 37
 - SQL procedure 49, 50
 - updating data files 35
- update types 126
- updating data
 - appending data 52
 - authorities 41
 - data files 35
 - deleting data records 47
 - directly from SAS 41
 - FSEDIT procedure 42, 43
 - FSVIEW procedure 42, 44
 - inserting data records 47
 - nulls 126
 - passwords 41
 - QUEST procedure 56
 - SAS/FSP procedures 42
 - SQL procedure 49
 - subsetting data 45
 - Version 7 and later 37
 - view descriptors for 44
 - WHERE clause and 45
- updating logical entries 65
- USER statement 108, 109

V

- variable names
 - based on data types 79
 - based on item names 79
 - created from schema item names 8
 - resetting to default 86
 - truncated 59
 - unique names 90
- variables
 - appending data and 52
 - creating databases 59

- deleting from database load 97
 - dropping from access descriptors 81
 - getting information about 24
 - level number for 99
 - list of information about 99
 - renaming 59, 64, 86
 - reviewing 24
 - terminology 23
 - Version 6 (or later) files
 - DBLOAD procedure and 59
 - Version 6 compatibility engine 35
 - Version 7 (or later) files 37
 - appending data to 55
 - DBLOAD procedure and 59
 - view descriptors 2, 19
 - access mode for creating 89
 - accessing databases without 56
 - appending data to 56
 - browsing and updating data 44
 - BY key 65
 - character item types 14
 - creating 70, 80
 - creating, from access descriptors 70
 - customized 92
 - effects of changing database definitions 116
 - efficient use of 78
 - example data 141
 - getting information about 24
 - in SQL procedure 49
 - including items from access descriptors 87
 - libref and member name 64
 - listing items and item information 84
 - mapping data between SAS and SYSTEM 2000 11
 - multiple 122
 - naming 103
 - ordering data 77
 - overriding data set option values 121
 - passwords for 68
 - selection criteria for 88
 - subsetting data for 88
 - VIEWDESC statement 103
 - VIEWDESC= option
 - PROC ACCESS statement 72
 - VIEWDESC= statement
 - DBLOAD procedure 103
- W**
- WHERE clause 15
 - browsing data and 45
 - connecting strings 133
 - NOT operator 132
 - not translatable to SYSTEM 2000 131
 - omitting, with connecting strings 135
 - selection criteria 133
 - SQL procedure 30, 50, 52
 - subsetting IF statement versus 45
 - translating to where-clause 130
 - where-clause versus 127, 133
 - WHERE command 45
 - WHERE statement 104
 - APPEND procedure 54
 - DBLOAD procedure 104
 - SAS/FSP procedures 45
 - subsetting input data 62
 - where-clause 14, 45, 72
 - AND operator 75
 - AT operator 76
 - binary operators 75
 - combining conditions 75
 - connecting strings 133
 - CONTAINS operator 75
 - HAS operator 76
 - NOT operator 76, 132
 - nulls in selection criteria 127
 - OR operator 75
 - position in database 76
 - processing order 76
 - record types 76
 - selection criteria 133
 - stored strings 136
 - syntax 72
 - ternary operators 75
 - translating WHERE clause to 130
 - unary operators 74
 - unmatched conditions 76
 - WHERE clause versus 127, 133
 - WHERE= data set option
 - appending data and 54
 - whole numbers 13



Gain Greater Insight into Your SAS[®] Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

 support.sas.com/bookstore
for additional books and resources.


THE POWER TO KNOW.

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 trademarks of their respective companies. © 2013 SAS Institute Inc. All rights reserved. S107969US.0613

