## Contents

*Changes and Enhancements*  

**PART I  SAS Guide to VSAM Processing: Overview and Techniques**  

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The Virtual Storage Access Method (VSAM)</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>SAS Options and Variables for VSAM Processing</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>Processing VSAM Data Sets in SAS Programs</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>Defining and Loading the Sample VSAM Data Sets</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>Processing an ESDS in a SAS Job</td>
<td>47</td>
</tr>
<tr>
<td>6</td>
<td>Processing a KSDS in a SAS Job</td>
<td>55</td>
</tr>
</tbody>
</table>

**PART II  SAS Guide to VSAM Processing: Detailed Description**  

**PART III  SAS Guide to VSAM Processing: Additional Information**  

**PART IV  SAS Guide to VSAM Processing: Appendices**
### Updating Records in a KSDS
- Page 65

### Erasing Records from a KSDS
- Page 66

### Combined Operations on a KSDS
- Page 67

#### Chapter 7 △ Processing an RRDS in a SAS Job
- Introduction: Page 73
- Special SAS Options Used with an RRDS: Page 74
- Reading Records from an RRDS: Page 74
- Adding Records to an RRDS: Page 78
- Updating Records in an RRDS: Page 80
- Erasing Records from an RRDS: Page 82
- Combined Operations on an RRDS: Page 82

#### Chapter 8 △ Using Alternate Indexes for VSAM Data Sets
- Introduction: Page 87
- Creating an Alternate Index for an ESDS: Page 87
- Creating an Alternate Index for an Existing KSDS: Page 89
- Calculating Record Size: Page 91

#### Chapter 9 △ Error-Handling Techniques and Error Messages
- Introduction: Page 93
- Physical Errors: Page 93
- Logical Errors: Page 94
- Error-Handling Techniques: Page 95
- Some Common Causes of Logical Errors: Page 97
- COBOL Status Key Values and VSAM Feedback Codes: Page 98

### P A R T 2  Appendices
- Page 101

#### Appendix 1 △ System Dependencies
- Introduction: Page 103
- CMS System Dependencies: Page 103

#### Appendix 2 △ Sample STUDENT Data Set
- Page 107

### Glossary
- Page 109

### Index
- Page 113
Changes and Enhancements

Beginning with Version 7, the VSE operating environment is not supported by the SAS System.
SAS Guide to VSAM Processing: Overview and Techniques

Chapter 1. The Virtual Storage Access Method (VSAM) 3
Chapter 2. SAS Options and Variables for VSAM Processing 15
Chapter 3. Processing VSAM Data Sets in SAS Programs 25
Chapter 4. Defining and Loading the Sample VSAM Data Sets 41
Chapter 5. Processing an ESDS in a SAS Job 47
Chapter 6. Processing a KSDS in a SAS Job 55
Chapter 7. Processing an RRDS in a SAS Job 73
Chapter 8. Using Alternate Indexes for VSAM Data Sets 87
Chapter 9. Error-Handling Techniques and Error Messages 93
Introduction

This chapter contains an overview of the Virtual Storage Access Method (VSAM). It presents basic VSAM terminology and concepts that you will need to process VSAM data sets. In particular, you will learn what access methods are and how they relate to data set organization. Then you will be introduced to the three types of access that VSAM provides to records in VSAM data sets. The access types are

- sequential
- direct
- skip sequential.

Each of these access types is fully described in this chapter.

You will learn about the three types of VSAM data sets. The organization of your VSAM data set determines how you can process it. The three types of data set organization are

- ESDS
- KSDS
- RRDS.

These data set types are introduced and discussed briefly in this chapter. Each data set type is described in detail in Chapter 5, “Processing an ESDS in a SAS Job,” on page 47, Chapter 6, “Processing a KSDS in a SAS Job,” on page 55, and Chapter 7, “Processing an RRDS in a SAS Job,” on page 73, respectively. The VSAM data set types are summarized in Table 1.1 on page 7.

In addition, five figures are presented to illustrate the concepts presented in this chapter. Figure 1.1 on page 7 demonstrates the organization of VSAM data sets. Figure
1.2 on page 9, Figure 1.3 on page 10, and Figure 1.4 on page 11 show the differences between an ESDS, a KSDS, and an RRDS. Finally, Figure 1.5 on page 13 illustrates a KSDS alternate index.

This chapter addresses the needs of the new VSAM user. If you are already familiar with the information that is presented here, you might want to proceed to Chapter 2, “SAS Options and Variables for VSAM Processing,” on page 15, which explains how to access VSAM data sets with the SAS System.

What is VSAM?

VSAM stands for Virtual Storage Access Method and is available under the CMS, OS/390, and VSE operating environments. Beginning with Version 7, VSE is not supported by the SAS System; therefore, VSE concerns are not described in this book.

VSAM is an IBM data access method that provides three ways to organize records in a disk data set. The three kinds of data set organization are
- ESDS (Entry-Sequenced Data Set)
- KSDS (Key-Sequenced Data Set)
- RRDS (Relative-Record Data Set).

VSAM allows three types of access to records in VSAM data sets. They are
- sequential
- direct
- skip sequential.

In addition, VSAM provides
- two direct access modes (addressed or keyed)
- two access entities (logical records and control intervals)
- two access directions (forward and backward)
- retrieval options (such as generic key and key greater-than-or-equal).

The SAS System supports all of these VSAM features, although not necessarily in all possible combinations. By specifying options in the INFILE statement in your SAS program, you can read, update, create, and erase records from VSAM data sets. See Table 3.1 on page 28 for a summary of the operations that the SAS System supports.

Access Methods

Access methods are software routines that control the data transfer between primary storage (main memory) and secondary storage devices. Secondary, or auxiliary storage, is independent of the computer’s memory, for example, storage on tape or disk. VSAM is designed specifically for use with disks. Because VSAM data set structure permits the use of both direct and sequential access types, you can select either the type or the combination of access types that best suits your specific application requirements.

Direct access means that you have the ability to read any data record in a data set directly, without reading preceding records in the data set. The terms direct and random are sometimes used interchangeably when referring to data organization, access methods, and storage devices. In this document, only direct is used, but you may find that random is used in other literature.

Access Methods and File Organization

Data stored on IBM disks can be organized in a number of ways, which are referred to as data set types. IBM software supports the following data set types:
The Virtual Storage Access Method (VSAM) PROVIDES A FUNCTIONAL EQUIVALENT FOR MOST OF THESE DATA SET ORGANIZATIONS, AND IT IS A MULTIFUNCTION, ALL-PURPOSE ACCESS METHOD. FOR INSTANCE,

- ESDS organization is the functional equivalent of physical sequential organization (PS).
- KSDS organization is the functional equivalent of indexed sequential organization (IS).
- RRDS organization is the functional equivalent of direct access organization (DA).

The kinds of data set organizations that you access with VSAM differ from some others in that they are device independent from the user's viewpoint and can be both sequentially and directly accessed. You access a record by addressing the record in terms of its displacement (in bytes) from the beginning of the data set, by its key, or by its record number.

The root of the VSAM access method is the VSAM catalog, which is a disk area for defining data sets and disk space and maintaining information about each VSAM data set. VSAM catalogs and data sets are created and managed with IBM Access Method Services (AMS), a multifunction service program.

### Types of VSAM Data Sets

There are three types of VSAM data sets. The main difference between the three data set types is the logical order in which data records are arranged in the data set. The following is a description of each type of VSAM data set:

- **ESDS** Entry-Sequenced Data Set — The record sequence is determined by the order in which the records are entered into the data set, without respect to the record contents. New records are stored at the end of the data set.

  An ESDS is appropriate for applications that do not require any particular ordering of the data by the record contents or for those that require time-ordered data. Applications that use a log or journal are well-suited for an ESDS data set structure.

- **KSDS** Key-Sequenced Data Set — The record sequence is determined by a key containing a unique value, such as an employee, invoice, or
transaction number. The key is a contiguous portion of the record and is defined when the data set is created. The record order is defined by the EBCDIC collating sequence of the key field contents.

A KSDS is always defined with a prime index that relates the record's key value to its relative location in the data set. VSAM uses the index to locate a record for retrieval and to locate a collating position for record insertion.

A KSDS is the most flexible approach for most applications since the record can be accessed directly, via the key field. Access is not dependent on the physical location of the record in the data set.

**RRDS**

Relative-Record Data Set — The data set is a string of fixed-length slots, each identified by a relative record number (RRN). Each slot can either contain a record or be empty. Records are stored and retrieved by the relative record number of the slot.

An RRDS is appropriate for many applications using fixed-length records or when the record number has a contextual meaning that can be used as a key.

Figure 1.1 on page 7 shows how the three types of VSAM data sets are organized. When a VSAM data set is created, it is defined in a cluster. A cluster encompasses the components of a VSAM data set. ESDS and RRDS clusters have only a data component. A KSDS cluster has a data component and an index component. The index relates each record's key to its location in the data set. VSAM uses the index to sequence and locate the records of a KSDS.

Table 1.1 on page 7 summarizes the differences between the three VSAM data set types.
**Figure 1.1** VSAM Data Set Organization: Data Components and Index Components

**Table 1.1** Comparison of VSAM Data Set Types

<table>
<thead>
<tr>
<th>Feature</th>
<th>ESDS</th>
<th>KSDS</th>
<th>RRDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential access is by</td>
<td>entry order</td>
<td>primary key order</td>
<td>RRN**</td>
</tr>
<tr>
<td>Direct access is by</td>
<td>RBA**</td>
<td>key RBA</td>
<td>RRN</td>
</tr>
<tr>
<td>Record format can be</td>
<td>fixed</td>
<td>fixed</td>
<td>fixed</td>
</tr>
<tr>
<td></td>
<td>variable</td>
<td>variable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>spanned</td>
<td>spanned</td>
<td></td>
</tr>
<tr>
<td>Changeable record length</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>New records added to</td>
<td>end of file</td>
<td>anywhere</td>
<td>RRN slot</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(if empty)</td>
</tr>
<tr>
<td>Embedded free space defined*</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Delete records and reuse space</td>
<td>no**</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Access through alternate index</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>
VSAM Record Structure and Organization

Records in VSAM data sets are grouped into control intervals, the unit of data transfer between main storage and secondary disk storage. Control intervals are continuous areas of direct access storage that VSAM uses for storing records and control information describing them. Although the size of control intervals varies from one data set to another, the size within a data set is fixed, either by VSAM or by the user (within VSAM-imposed restrictions). If VSAM chooses the size, it does so based on the DASD type, record size, and smallest amount of virtual storage space that user applications make available for I/O buffers. A spanned record is one that exceeds the established control interval size by spanning one or more control interval boundaries. Spanned records are permitted in an ESDS and a KSDS but not in an RRDS.

Control intervals are grouped into control areas. Control areas are the units of a data set that VSAM preformats as records are added to the data set. VSAM fixes the number of control intervals for each control area. (Refer to Figure 1.2 on page 9, Figure 1.3 on page 10, and Figure 1.4 on page 11 for a depiction of the control interval format used by each of the data set types.) In addition, KSDS control areas are used for distributing free space throughout the data set, as a percent of control intervals per control area.
Figure 1.2  ESDS Control Intervals and Control Areas

Logical records can vary in length, but the length cannot change once the record is written. All free space is at the end of the file.

- CI - control intervals
- LR - logical record
- Free Space

Control area 1  Control area 2  •  •  •  Control area n
Figure 1.3 KSDS Control Intervals and Control Areas

Logical records can vary in length. Free space is distributed throughout the file for inserting, deleting, lengthening, or shortening records.

- Control intervals
- Control areas
CI - control information
LR - logical record
The Virtual Storage Access Method (VSAM) / Direct Access

VSAM Data Access Types

Access to VSAM data sets can be either sequential, direct (random), or a combination of both. Records can be retrieved in either a forward or backward direction.

Sequential Access

In sequential access, a series of records is retrieved in sequence. Sequence has a different meaning for each of the three VSAM data set organizations.

- In an ESDS, sequential access means that a record is retrieved by its entry sequence.
- In a KSDS, sequential access means that a record is retrieved by its key sequence.
- In an RRDS, sequential access means that a record is retrieved by its relative record sequence.

In all three cases, a record is located by its position relative to the last record accessed.

Direct Access

With direct access, data storage or retrieval depends only on the location of the record and not on a reference to records previously accessed. Each record is stored or retrieved directly, according to its logical address (its key or relative record number, RRN) or its address relative to the beginning of the data set (relative byte address, or
RBA). Thus, there are two direct access modes: *keyed* by key or relative record number, and *addressed* by relative byte address.

**Keyed direct access**

In keyed direct access, records are retrieved or stored by either
- an index that relates the record’s key to its relative location in the data set
- a relative record number (RRN) that identifies the record wanted. The RRN is relative to the first record in the data set.

The SAS System supports keyed access to logical records in both KSDS and RRDS data sets. Keyed access to data records in KSDS data sets is by key; in RRDS data sets, keyed access is by the relative record number.

**Addressed direct access**

In addressed direct access, the entire data set is treated as a continuous stream of bytes. A record is retrieved and stored directly by its address relative to the beginning of the data set (relative byte address, or RBA), which is dependent on the record’s location relative to records previously accessed. The SAS System supports addressed access to logical records in ESDS and KSDS data sets. It also supports addressed access (read only) to control intervals in all three data set types.

**Keyed direct access with an alternate index**

An alternate key index, commonly called an alternate index (AIX), provides another way to access a VSAM data set. The advantage of an alternate index is that you effectively reorganize the data set instead of keeping separate copies organized in different ways for different applications. For example, suppose you have a KSDS with the employee number as the prime key. By building alternate indexes using employee names and department numbers, you can access the same data set in three ways: by employee name, by employee number, or by department number. The alternate key does not have to be unique; that is, there can be more than one record with the same alternate key.

Figure 1.5 on page 13 illustrates an alternate index with nonunique keys over a KSDS. The base cluster records are sequenced by employee number, which is the prime key. The alternate index records are sequenced by department number, which is the alternate key. Each alternate index data record points to the prime key (employee number) in the base cluster. Note that because the alternate keys are nonunique there can be multiple base records with the same department number.
Alternate indexes can be built over a KSDS or an ESDS. You define and build an alternate index using the IBM utility program Access Method Services (AMS). The data set over which an alternate index is built is called the base cluster. The alternate key can be any field having a fixed length and a fixed position within each record. The alternate index itself is a KSDS. The data component of an alternate index contains the alternate key, followed by a pointer to the appropriate record(s) in the base cluster. In a KSDS, the pointer is the prime key; in an ESDS, the pointer is the RBA of the base record(s).

A path logically relates a base cluster and one of its alternate indexes. You define and name a path to access the base cluster records through a specific alternate index with AMS. Refer to Chapter 8, “Using Alternate Indexes for VSAM Data Sets,” on page 87 and the appropriate IBM documentation for more information on defining alternate indexes.
**Skip Sequential Access**

A combination of both direct and sequential access can be used in a two-step process called skip sequential access. Use keyed direct access to find a starting point. Once the initial record is obtained, additional records are retrieved sequentially. Skip sequential processing can be used with a KSDS, RRDS, and, if it has an alternate index, an ESDS.

Skip sequential processing is useful when

- performance is important. Processing skip sequentially can improve performance and reduce overhead since a simple sequential retrieval is faster than direct retrieval.
- you know the key, RBA, or RRN of the first record you want but do not know the key, RBA, or RRN of the subsequent records.
- you want to process the data set sequentially, starting at a record other than the first.
Introduction

The SAS System provides access to VSAM data sets through the DATA step INFILE, INPUT, FILE, and PUT statements. A group of special VSAM options for the INFILE and FILE statements is used along with standard INFILE and FILE options to read and write VSAM data sets.

This chapter describes the global SAS system options and the automatic SAS variables that are set when you process VSAM data sets. Also, you will become familiar with both standard and special SAS INFILE options. “Special SAS Options for VSAM” on page 19 presents a complete, descriptive list of each INFILE option available for processing your VSAM data sets. This list is very helpful, so you may want to refer to it as you read later chapters in this manual.

Refer to Chapter 3, “Processing VSAM Data Sets in SAS Programs,” on page 25 and the appropriate reference chapter for the VSAM data set that you want to access for information about using these options to process VSAM data.

SAS System Options

There are three global SAS system options used with VSAM data set processing:

VSAMREAD|NOVSAMREAD
enables or disables the reading of VSAM data sets.

VSAMUPDATE|NOVSAMUPDATE
enables or disables the updating of VSAM data sets by modifying or erasing existing records or by adding new records. VSAMUPDATE implies VSAMREAD.

VSAMLOAD|NOVSAMLOAD
enables or disables the loading of records into a new VSAM data set.

The SAS System is distributed with the following default values:

- VSAMREAD
Your site administrator may have changed the default settings. Use the OPTIONS procedure to check the settings of these and any other global SAS system options. Your site may prevent you from overriding these options.

### SAS Automatic Variables

The following SAS automatic variables are created when a VSAM data set is accessed and set when records in the data set are accessed. (Records are accessed when an INPUT or a PUT statement that reads or writes a record executes.) These automatic variables are not added to any SAS data sets created by a DATA step.

- `_RBA_` contains the RBA of the last record accessed.
- `_IORC_` is set to the VSAM input/output return code.
- `_FDBK_` is set to the VSAM feedback code.
- `_RRN_` contains the RRN of the last RRDS record accessed. This variable is not created for ESDS and KSDS data sets.

### Standard SAS INFILE Options

External data sets can be standard or nonstandard. VSAM data sets are nonstandard external data sets. The following SAS INFILE statement options can be used with any external data set, including VSAM data sets. Descriptions of options designed especially for VSAM processing are in the next section of this chapter. For more information on standard and nonstandard SAS options, refer to the `INFILE Statement` in the *SAS Language Reference: Dictionary, Volumes 1 and 2*.

- **BLKSIZE=value**
  - specifies the block size of the input data set.
- **COLUMN=variable**
  - defines a variable that the SAS System sets to the column location of the pointer.
- **DELIMITER=delimiters**
  - specifies a delimiter other than a blank for list input.
- **DSD**
  - changes the way delimiters are treated when using list input; this option enables you to read delimiters as characters within quoted strings.
- **END=variable**
  - defines a variable, whose name you supply, that the SAS System sets to 1 when the current record is the last in the input data set. Until SAS processes the last record, the value of the END= variable is 0. You cannot use the END= option with direct access.
- **EOF=label**
  - specifies a statement label as the object of an implicit GO TO when the INFILE statement reaches end-of-file. When an INPUT statement attempts to read from a data set that has no more records, SAS moves execution to the statement label indicated. The EOF= option is ignored with direct access.
EOV=variable
   specifies a variable, whose name you supply, that the SAS System sets to 1 when the first record in a data set in a series of concatenated data sets is read. The variable is set only after SAS encounters the next data set.

EXPANDTABS|NOEXPANDTABS
   specifies whether to expand tab characters to the standard tab setting.

FILENAME=variable
   defines a variable, whose name you supply, that the SAS System sets to the value of the physical name of the currently open input data set.

FILEVAR=variable
   defines a variable whose name you supply and whose change in value causes the INFILE statement to close the current input data set and open a new one. This option is not available under CMS because the physical path name of a VSAM file is defined with a DLBL command.

FIRSTOBS=record-number
   indicates you want to begin reading the input data set at the record number specified rather than beginning with the first record.

FLOWOVER
   specifies the action to be taken if the INPUT statement reads past the end of the current record. When FLOWOVER is in effect, the SAS System reads a new record, and the INPUT statement continues reading data from column one to the first blank in the new record.

_INFILE_=variable
   names a variable that the SAS System will use to reference the contents of the current input buffer of this INFILE statement. Like automatic variables, the _INFILE_= variable is not written to the data set.

LENGTH=variable
   defines a variable, whose name you supply, that the SAS System sets to the length of the current line.

LINE=variable
   defines a variable, whose name you supply, that the SAS System sets to the line location of the INPUT or PUT pointer. The LINE= option cannot be used with direct access.

LINESIZE=value
LS=value
   limits the record length available to the INPUT statement when you do not want to read the entire record. The LINESIZE= value specifies the maximum record length that is available to the SAS program. If a LINESIZE= value is not specified, the default is the maximum record length specified when the VSAM data set was defined.
   If LINESIZE= is shorter than the VSAM maximum record length, the record is truncated to the specified LINESIZE= value.
   If an INPUT statement attempts to read past the column specified by LINESIZE=, the action taken depends on which of the FLOWOVER, MISSOVER, and STOPOVER options is in effect. (By default, the MISSOVER option is in effect when reading VSAM data sets.)
   If a PUT statement attempts to write a record longer than the value specified by LINESIZE=, the action taken depends on which of the FLOWOVER, MISSOVER, and STOPOVER options is in effect. (By default, the STOPOVER option is in effect when writing VSAM data sets.)
LRECL=value
specifies the logical record length of the file. If you do not specify an option, SAS chooses a value based on the operating environment's file characteristics.

MISSOVER
prevents a SAS program from going to a new input line if it does not find values in the current line for all the INPUT statement variables. When an INPUT statement reaches the end of the current record, values that are expected but not found are set to missing.

N=available-lines
specifies the number of lines you want available to the input pointer. The N= option cannot be used with direct access since, by definition, direct access gets only one line at a time.

OBS=record-number
specifies the record number of the last record you want to read from an input file that is being read sequentially. Counting begins at the value set in the FIRSTOBS= option, if specified.

PAD | NOPAD
specifies whether records read from an external data set are padded with blanks up to the length specified in the LRECL= option.

PRINT | NOPRINT
specifies whether the input data set contains carriage-control characters.

RECFM=record-format
specifies the record format of the input data set.

SCANOVER
specifies that the INPUT statement scan the input records until the character string specified in the @’character string’ expression (on the INPUT statement) is found.

SHAREBUFFERS
specifies that the FILE statement and the INFILE statement share the same buffer.

START=variable
defines a variable whose name you supply and whose value is used as the first column number of the record that the _INFILE_ argument of the PUT statement is to write.

STOPOVER
stops processing the DATA step when an INPUT statement reaches the end of the current record without finding values for all variables in the statement.

When the STOPOVER option is specified and an input line does not contain the expected number of values, the SAS System sets _ERROR_ to 1, stops building the data set as though a STOP statement had executed, and prints the incomplete data line.

TRUNCOVER
overrides the default action of the INPUT statement when an input record is not as long as expected by the INPUT statement.

UNBUFFERED
tells the SAS System not to perform a buffered read.
Special SAS Options for VSAM

The special SAS options for VSAM data sets are specified in the INFILE statement except when you load a new VSAM data set with initial records. You must use the FILE statement when you load a new VSAM data set. For more information on FILE statement options for VSAM data sets, refer to “Loading Records into a VSAM Data Set” on page 45.

The special SAS options for VSAM are shown below. All the options except those described in Table 2.1 on page 19 can be used with all three VSAM data set types.

Table 2.1 Special SAS Options for Selected VSAM Data Sets

<table>
<thead>
<tr>
<th>Option</th>
<th>ESDS through AIX</th>
<th>ESDS</th>
<th>KSDS</th>
<th>RRDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERASE=</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>GENKEY</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEY=</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEYGE</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEYLEN=</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEYPOS=</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RBA=</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RRN=</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SEQUENTIAL</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SKIP</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

BACKWARD
BKWD

instructs the SAS System to read a VSAM data set backwards. You can use the BACKWARD option only when you are reading the VSAM data set sequentially.

BUFND=integer

specifies the number of data buffers for a VSAM input data set. If BUFND= is not specified, VSAM provides a default value. For sequential processing, two data buffers are usually sufficient. If your VSAM application’s performance is slow, it might indicate that you should increase the number of data buffers that are specified by BUFND=. The systems programming staff at your installation can help you determine what values to assign to BUFND=.

BUFNI=integer

specifies the number of index buffers for a VSAM data set. If BUFNI= is not specified, VSAM provides a default value. For sequential processing, one index buffer is usually sufficient. If your VSAM application’s performance is slow, it might indicate that you should increase the number of data buffers specified by BUFNI=. The systems programming staff at your installation can help you determine what value to assign to BUFNI=.
CONTROLINTERVAL
CTLINTV
CNV

specifies that you want to read VSAM control intervals rather than logical records. When you specify the CONTROLINTERVAL option for a password-protected VSAM data set, you must give a control-interval-access or higher-level password with the PASSWD= option.

Control intervals cannot be updated or erased. Control interval access is typically used only for diagnostic applications or for reading a VSAM catalog.

ERASE=variable

defines a numeric SAS variable that you must set when you want to erase a VSAM record. The ERASE= option must be specified to erase a VSAM record.

The record is erased when you set the ERASE= variable to a value of 1 before a PUT statement for the output data set executes. When you set the ERASE= variable to a value of 0, the record is updated (instead of erased) when the PUT statement executes. This is the default action if ERASE= is not specified.

After a record is erased, the ERASE= variable is automatically reset to 0. Therefore, you must reset the ERASE= variable to 1 in order to erase another record. This prevents the inadvertent deletion of a series of records.

This option is valid only for KSDS and RRDS records. There is a VSAM restriction that records cannot be erased from an ESDS.

FEEDBACK=variable
FDBK=variable

defines a numeric SAS variable that the SAS System sets to the VSAM logical error code when a logical error occurs. FEEDBACK= is similar to the _FDBK_ automatic variable, but it is more flexible and less likely to allow VSAM logical errors to go unnoticed. When the SAS System sets the FEEDBACK= variable, you must reset it to 0 to continue processing after a logical error. See Chapter 9, “Error-Handling Techniques and Error Messages,” on page 93 for more information.

GENKEY

specifies generic-key processing for a KSDS. When GENKEY is specified, SAS programs treat the KEY= variable as the leading portion of a record’s key. The SAS System retrieves the first record whose key matches the generic key, unless you also specify skip sequential processing. Use this option if you plan to retrieve a series of KSDS records that have the same leading key field, or if you know only the leading portion of a particular key. The GENKEY option applies to all records read from the data set in the DATA step; that is, you cannot turn GENKEY on and off. Changing the value of the KEY= variable indicates another generic-key retrieval request.

When you specify both the GENKEY and the SKIP options, the SAS System retrieves the first record containing the matching partial key and then reads the following records sequentially. Access is sequential after the first key until you change the value of the KEY= variable, which indicates another direct-access, generic-key retrieval request. Assigning a new value that equals the previous value is not regarded as a change. To perform repeated direct-access, generic-key retrievals with the same KEY= value, you must clear and reassign the KEY= variable after each retrieval.

KEY=variable
KEY=(list of variables)

indicates that keyed direct access is to be used to retrieve records from a KSDS or an ESDS that was accessed through an alternate index. The KEY= option specifies either one variable or a list of variables that provides the key of the record to be read. You can construct a key up to 256 characters long by defining a list of up to
16 character variables. The SAS System builds the key by concatenating the variables; blanks are not trimmed. The key is extended with blanks on the right until it reaches the full key length set during creation of the VSAM data set with AMS/IDCAMS.

Unless used with the GENKEY option, the key value that is passed to VSAM is either padded with blanks or truncated, as necessary, to equal the key length defined when the KSDS is created.

**KEYGE**

specifies that retrieval requests with the KEY= option are for any record whose key is equal to or greater than the key specified by the KEY= option. This approximate key retrieval is useful when the exact record key is not known. The KEYGE option applies to all records read from the data set in that DATA step; that is, you cannot turn KEYGE on and off.

**KEYLEN=variable**

specifies a numeric SAS variable that, when used with GENKEY, specifies the length of the key to be compared to the keys in the data set. That is, the variable’s value is the number of generic key characters passed to VSAM. If you specify GENKEY without the KEYLEN= option, the generic-key length is the KEY= variable length (or the sum of the KEY= variable lengths, if a list is specified).

The SAS System sets the variable specified by the KEYLEN= option to the actual key length defined in the cluster before the DATA step executes. The KEYLEN= option can be used to read KSDS keys without any advance knowledge of the key length. Assign the value of the KEYLEN= variable to a different variable if you also intend to set the KEYLEN= variable for generic key processing. You may need to name the variable in a RETAIN statement if you need this initial value after the first execution of the DATA step.

**KEYPOS=variable**

specifies a numeric SAS variable that the SAS System sets to the position of the key field. This option allows KSDS keys to be read without advance knowledge of the key position. The variable is set to the column number, not the offset, which is one less than the column number.

When you use the KEYLEN= and the KEYPOS= options together, it is possible to read KSDS keys without knowing either the key position or length in advance. The SAS variables that you specify with the KEYLEN= and KEYPOS= options should not be present in any SAS data set used as input to the DATA step.

**PASSWD=password**

gives the appropriate password for a data set that has VSAM password protection. The password is replaced with Xs on the SAS log. The appropriate password is

- a read (or higher-level) password for a data set that you are reading only.
- an update (or higher-level) password for a data set that you are updating or loading.
- a control interval (or higher-level) password to read a data set’s control intervals directly.

**RBA=variable**

defines a numeric variable that you set to the relative byte address (RBA) of the data record (or control interval) to be read. The RBA= option indicates that addressed direct access is to be used for record retrieval from an ESDS or a KSDS. The RBA= option can also be used to access the control intervals in an RRDS if the CONTROLINTERVAL option is specified.
RC4STOP
stops the DATA step from executing when opening a VSAM data set results in a
return code of 4. The SAS System normally treats a return code of 4 from OPEN
as recoverable and continues processing. This option is ignored under OS/390.

READPW=password
is a synonym for the PASSWD= option.

RECORDS=variable
defines a numeric variable that the SAS System sets to the number of logical
records in the VSAM data set you are reading.

RESET
specifies that the VSAM data set is to be reset to empty (no records) when it is
opened. The RESET option applies only to loading a VSAM data set that has been
defined with the VSAM option “REUSE.” Specify this option to use a VSAM data
set as a work data set by reloading it in the DATA step. This option cannot be
used if the data set has an alternate index.

RRN=variable
defines a numeric variable that you set to the relative record number (RRN) of the
record to be read or written. This option indicates that keyed direct access is to be
used for record storage and retrieval and is appropriate only for an RRDS.

SEQUENTIAL
SEQ
specifies sequential record retrieval when either the RBA= (for an ESDS) or the
RRN= (for an RRDS) direct access option indicates direct record storage for the
PUT statement(s).

The SEQUENTIAL option is necessary only when adding new records after
sequentially reading existing records in an ESDS or an RRDS.

SKIP
indicates that skip sequential access is to be used for record retrieval. Skip
sequential access finds an initial record with keyed direct access and then
retrieves records from that point on with sequential access. An unchanged KEY=
or RRN= value indicates that subsequent records are to be retrieved sequentially.
The SKIP option can be used for a KSDS or an RRDS.

UPDATE=variable
defines a numeric variable that tells the SAS System that not every record read is
to be updated when you are reading and writing records in a VSAM data set.
When you have both an INFILE and a FILE statement referencing the same
VSAM data set, records are retrieved for update by default.

In most cases, when a record is retrieved for update, no user, including you, can
access that particular record or any other records in the same control interval
until you free the record by executing a PUT or an INPUT statement for the data
set. The UPDATE= option is used to avoid user lockout when only a few of many
records read need to be updated. When you set the UPDATE= variable to
  □ a value of 1 before an INPUT statement executes, the record is retrieved for
    update. This is the default if UPDATE= is not specified.
  □ 0 before the INPUT statement executes, the record is not retrieved for update.

VSAM
indicates that the fileref points to a VSAM nonstandard, external data set. On
CMS, the VSAM option must immediately follow the fileref in the INFILE and
FILE statements. On OS/390, it is optional except when you bypass the VSAM
catalog to determine the volume location of the VSAM component or cluster and
you code the AMP=('AMORG') parameter in the JCL that defines the VSAM component or cluster.

**WRITEPW=password**

is a synonym for the PASSWD= option.

---

**Using the Special SAS Options**

The special SAS options used for processing VSAM data sets fall into three functional categories:

- options that describe the characteristics of the VSAM data set and how the SAS System is to process it
- record retrieval options (options processed by the INPUT statement)
- record storage options (options processed by the PUT statement).

Refer to Table 5.1 on page 48, Table 6.1 on page 56, and Table 7.1 on page 74 for information on how each special SAS option functions when it is used with the three VSAM data set types.

You specify the special SAS options in the INFILE statement except when you load a new VSAM data set with initial records. When you load a new VSAM data set, you specify the special SAS options in the FILE statement.

Note the following three important points concerning the INFILE statement with VSAM data sets:

1. Because VSAM options are specified in the INFILE statement, this statement has the extra function of setting up how an operation is to be performed.
2. Because of this setup function, the INFILE statement is sometimes used without a corresponding INPUT statement.
3. The INFILE statement is not used to load records. Thus, loading records is treated as a special case and is discussed separately in Chapter 4, “Defining and Loading the Sample VSAM Data Sets,” on page 41.

The syntax for using the special options in the INFILE statement is

```
INFILE data-set-specification <options>;
```

where **data-set-specification** is a SAS fileref or a physical filename on OS/390 and an operating system fileref on CMS, and **options** are a combination of standard and special INFILE statement options.

A number of the SAS VSAM options specify SAS variables that contain values that control VSAM operations. The settings of such variables are not constants. They can be changed within a DATA step with SAS assignment statements. Variables that are specified by the ERASE= and FEEDBACK= options **must** be reset in a DATA step.

The variables specified by SAS VSAM options are not automatically added to any output SAS data set.
VSAM Option for the FILENAME Statement

On OS/390 when you load a new VSAM data set, you can specify the VSAM option RECORG= in the FILENAME statement.

RECORG= record-organization
specifies the organization of records in a new VSAM data set. Use this option only if SMS is active. Valid values are

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KS</td>
<td>VSAM key-sequenced data set</td>
</tr>
<tr>
<td>ES</td>
<td>VSAM entry-sequenced data set</td>
</tr>
<tr>
<td>RR</td>
<td>VSAM relative-record data set</td>
</tr>
<tr>
<td>LS</td>
<td>VSAM linear-space data set</td>
</tr>
</tbody>
</table>
Introduction

This chapter is divided into two parts. The first part provides basic information on processing VSAM data sets in SAS programs. It contains instructions for determining the type of existing VSAM data sets on your site’s operating system and describes the operations that you can perform on VSAM data sets. Note that program statements and examples are not presented in this chapter. Instead, examples of the operations are given in the separate chapters for each data set type (ESDS, KSDS, and RRDS). Options and operational details that are specific to a given data set type are explained and illustrated in the appropriate chapter.

The second part contains examples of using VSAM data in SAS programs. Examples are given that show you how to print the contents of a VSAM data set, generate reports using PROC MEANS, and write a windowing application to update VSAM data.
Determining the Type of an Existing Data Set

You can determine the type of an existing VSAM data set by examining the SAS log after you load the data set for processing.

If the data set is an ESDS, the log displays the note

Type=NONINDEXED

If the data set is a KSDS, the log displays the note

Type=INDEXED

If the data set is an RRDS, the log displays the note

Type=NUMBERED

Referring to VSAM Data Sets

On OS/390, you can use a SAS fileref or the DDname as a convenient way of referring to a VSAM data set in a SAS language statement or command. The first time the DDname is used in a SAS statement or procedure, SAS assigns it as a fileref for the VSAM data set. Alternatively, you can use the FILENAME statement to explicitly associate a fileref with a VSAM data set. The FILE statement later uses the fileref rather than the data set name to refer to the data set.

On CMS, the DOS DLBL statements that are issued to define the VSAM data set to the CMS operating system make the correct logical connection between the SAS System and VSAM. You must use the DOS DLBL fileref to refer to VSAM data sets on CMS.

Operations on VSAM Data Sets in SAS Programs

SAS programs handle VSAM data sets the same as any external data set. An external data set is one that is not a SAS data set. For instance,

□ the VSAM data set can be read in a DATA step. Information from the VSAM data set can be used to create a SAS data set if appropriate for the application.
□ the VSAM data set can be updated in a DATA step by adding new records or by modifying or erasing existing records.
□ a SAS data set can be created. You can manipulate this data set with SAS DATA steps or procedures and then use it to update the VSAM data set in a subsequent DATA step.
□ you can use a DATA step to generate a SAS data view of the VSAM data set.

Figure 3.1 on page 27 illustrates a typical SAS DATA step processing a VSAM data set. A VSAM external data set is shown as input to a SAS DATA step. Only the INFILE statement with some of the most common special SAS options and the FILE statement are shown. Notice that both the INFILE and FILE statements specify the VSAM option and the fileref that refers to the VSAM data set. The RBA=, KEY=, and RRN= direct access variables are shown as INFILE statement options that depend on whether the VSAM data set is an ESDS, a KSDS, or an RRDS. (You do not need to specify a direct access option to process the data set sequentially.) Notice also that the FEEDBACK= variable is specified in the INFILE statement. Remember that you would need both an INPUT and a PUT statement to read and update the VSAM data set.
You can perform five general types of operations on VSAM data sets in SAS programs. You can

- **read** records from an existing data set. All VSAM data set types can be read both sequentially and with one or more direct access modes.
- **add** new records to an existing VSAM data set.
- **update** an existing record by retrieving, modifying, and then writing it back to the data set. Note that the record must be retrieved before being updated.
- **erase** an existing record from an RRDS or a KSDS. The record must be retrieved before erasing it. Records cannot be erased from an ESDS.
- **load** new records into a new VSAM data set. This operation is discussed separately in Chapter 4, “Defining and Loading the Sample VSAM Data Sets,” on page 41.

When you perform these operations, you can use certain types of access with each data set type. Refer to Table 3.1 on page 28 for an outline of this information. Note that VSAM provides both sequential and some form of direct access for each data set type.
Table 3.1  Supported VSAM Operations and Access Types

<table>
<thead>
<tr>
<th>Operation</th>
<th>Access Type</th>
<th>ESDS</th>
<th>KSDS</th>
<th>RRDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Sequential:</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>Direct access by:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>key</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>generic key</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>RBA</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>RRN</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>Skip sequential:</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Update</td>
<td>Sequential:</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>Direct access by:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>key</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>RBA</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>RRN</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Add/Load</td>
<td>Sequential:</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>Direct access by:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>key</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>RBA</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>RRN</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Erase</td>
<td>Sequential:</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>Direct access by:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>key</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>RBA</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>RRN</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

Reading Records from a VSAM Data Set

You must specify either the VSAMREAD or the VSAMUPDATE global SAS system option in order to read VSAM data sets. Records can be read with sequential access, direct access, or with a combination of both by using skip sequential access. By default, read access is sequential.

Options in the INFILE statement specify how the read operation is to be performed. Chapter 5, “Processing an ESDS in a SAS Job,” on page 47, Chapter 6, “Processing a KSDS in a SAS Job,” on page 55, and Chapter 7, “Processing an RRDS in a SAS Job,” on page 73 describe reading data from each of the three VSAM data set types. In these chapters, you will find sections discussing the options that you can specify to retrieve records from the particular VSAM data set type that you are using.

Sequential Access

Sequential access means that a record is retrieved by its
- entry sequence in an ESDS
key sequence in a KSDS
relative record sequence in an RRDS.

Sequential record retrieval depends on the location of the previously retrieved record.
By default, the SAS System processes VSAM and other data sets sequentially. When access is sequential, the SAS System performs a standard look-ahead read. When the SAS System encounters the end of the data set while processing sequentially, the END= variable, if specified, is set to 1.

Direct Access

Access is direct if one of the direct access options (KEY=, RRN=, or RBA=) is specified in the INFILE statement.

When you access a VSAM data set directly in a SAS job, the standard look-ahead read used with sequential access is inhibited. Therefore, the END= INFILE statement option (if specified) is ignored with direct access because the END= variable is never set to 1. This means there is no automatic mechanism to end a DATA step that directly accesses a VSAM data set. Instead, you must end the DATA step explicitly. You can use the standard INFILE statement option EOF= and a STOP or SET statement. Use a STOP statement to end the DATA step when you are processing the VSAM data set with direct access; use a SET statement when you are processing an existing SAS data set against the VSAM data set.

Skip Sequential Access

Skip sequential access is a two-step process that combines both direct and sequential access. Once the initial record is located with keyed direct access, subsequent records are retrieved sequentially. Skip sequential processing can be used with a KSDS, an RRDS, and an ESDS with an alternate index. The SKIP option in the INFILE statement indicates skip sequential access. When you are processing a VSAM data set skip sequentially, you must explicitly end the DATA step with either a STOP statement or, if you are processing a SAS data set against the VSAM data set, with a SET statement.

Adding Records to a VSAM Data Set

When you add records to an existing VSAM data set, you must
1 specify the SAS system option VSAMUPDATE.
2 include both an INFILE statement and a FILE statement with the same fileref and the VSAM option in the DATA step. Specify any other options in the INFILE statement, which must precede the FILE statement.

Note: Because VSAM options are specified in the INFILE statement, this statement has the extra function of setting up how an operation is to be performed. Because of this setup function, the INFILE statement is sometimes used without a corresponding INPUT statement.

Ordinarily, the INFILE statement identifies an external data set to be read by an INPUT statement. However, when you add new records to an existing VSAM data set, you can use the INFILE statement without a corresponding INPUT statement (that is, without reading a record) because VSAM knows where to put the new records. For example,

- in an ESDS, records are added in entry order. Therefore, new records are always added with sequential access to the end of the data set.
in a KSDS, records are added with direct access. KSDS records are ordered by the prime key, which is part of the record itself.

in an RRDS, records are added with direct access. Because records are added by specifying the RRN, the RRN of a new record must be given via the value of the RRN= variable.

To add a new RRDS record, you must set the RRN= variable to the value of an empty relative record slot (number) before the PUT statement executes.

---

**Updating Records in VSAM Data Sets**

Performing an update operation involves both input access (since the record must be read first) and output access (since you update by writing to the data set). Input access for update can be either sequential or direct. Output access is sequential unless one of the direct access variables (KEY=, RRN=, or RBA=) is specified. When you update an existing record in a VSAM data set, you must

1. specify the SAS system option VSAMUPDATE.
2. include both an INFILE and a FILE statement with the same fileref and the VSAM option in the DATA step. Specify all other options in the INFILE statement, which must precede the FILE statement.
3. use an INPUT statement to read the record that is being modified.
4. use the PUT statement to write the complete record. An INPUT statement brings the record into the INPUT buffer but does not copy it to the PUT buffer. This allows you to change the record easily.

There are two common ways of writing the record with the PUT statement:

- Build the complete record by specifying all fields with the PUT statement. This method may be best when many of the fields need updating or when the updated record is shorter than the existing record because it avoids the problem of trying to eliminate or “blank out” the unwanted fields.
- Copy the input record to the output buffer (with PUT _INFILE_) and overlay selected fields in the copy. This method may be best when relatively few fields need to be updated.

The latter method is the easiest for most applications. The following statement copies the last record read into the PUT buffer and overlays the information starting in columns 10 and 30 with the values in NEWDATA1 and NEWDATA2:

```
PUT @ 1 _INFILE_
  @ 10 NEWDATA1
  @ 30 NEWDATA2;
```

**Limitations on Updating**

To maintain data integrity in multiple update situations under OS/390, VSAM uses operating system facilities to protect the data. However, when either SHROPTIONS(3 X) or (4 X) is specified (full sharing by any number of users) and records in the same control area are updated simultaneously, some of the updates may be lost (X is any value). When SHROPTIONS(3 X) is used, each user is responsible for maintaining both read and write integrity for the data the program accesses.

SHROPTIONS(4 X) requires your program to use the ENQ and DEQ macros to maintain data integrity while sharing the data set. For more information on using ENQ and DEQ, see the appropriate IBM documentation.

In order to update a VSAM data set under CMS SAS, you must have exclusive write access to the minidisk (or minidisks) that contain the data set and catalogs; otherwise,
the integrity of the data is jeopardized. If you intend to update a VSAM data set under CMS SAS, make sure that you do not link to any of the VSAM minidisks in MW (multiple write) mode.

**Using the UPDATE= Option**

Sometimes a program accessing a VSAM data set reads many records but updates only a few of the records. When a record is retrieved for update, no other user, including you, can access that particular record or any other records in the same control interval until you release the record by executing another PUT or an INPUT statement for the data set. (This is significant when a VSAM data set is simultaneously accessed by other users or by an online system such as CICS.) Use the UPDATE= option in the INFILE statement to avoid user lockout when only a few of the records that are retrieved need to be updated.

The UPDATE= option specifies a numeric SAS variable that indicates whether a record is to be read only or updated.

- When you set the UPDATE= variable to a value of 1 before an INPUT statement executes, the record is retrieved for update. This is the same action taken if the UPDATE= option is not specified.
- When you set the UPDATE= variable to a value of 0 before an INPUT statement executes, the record is not retrieved for update.

If you retrieve a record with the UPDATE= variable set to 0 (that is, the record is not retrieved for update) and then decide that you do want to update the record, reset the UPDATE= variable to 1, retrieve the record again, and then update the record. This is possible only with direct access. If you are reading the data set sequentially, you must keep track of the records that you want to update (use a SAS data set for this) and read them for update in a subsequent DATA step.

**Erasing Records from a VSAM Data Set**

Erasing a record involves both input access (because the record must be read first) and output access. You can erase records from a KSDS or an RRDS. The record must be retrieved before you can erase it, and you must specify the global SAS system option VSAMUPDATE. There is a VSAM-imposed restriction that ESDS records cannot be erased.

You must use an INFILE statement and an INPUT statement to read the record and a FILE statement and a PUT statement to erase the record from a VSAM data set. Of course, the INFILE and FILE statements must have the same fileref; that is, they must reference the same data set. You must use the ERASE= option in the INFILE statement to specify a numeric SAS variable that tells the SAS System whether or not a record is to be erased.

- When you set the ERASE= variable to a value of 1 before a PUT statement for the data set executes, the record is erased.

  After the PUT statement executes, the ERASE= variable is automatically reset to 0. Therefore, you must set it to 1 again to erase another record. This prevents the inadvertent deletion of a series of records.

- When you set the ERASE= variable to a value of 0 before a PUT statement for the data set executes, the record is updated with the data specified instead of being erased. This is the default action that is taken if the ERASE= option is not specified.
Combined Operations

You may want to perform more than one operation in one DATA step. For example, perhaps you want to read some records, update other records, and add new records all in one DATA step. Regardless of the number of different operations, only one pair of INFILE and FILE statements for the data set is needed during the DATA step. Specify all the options you may need for processing the data set in its INFILE statement.

In a DATA step that combines operations, the SAS System determines whether you want to update existing records or add new records as described below.

When you do not have an INPUT statement that is associated with the INFILE statement (because you are adding records without reading from the data set), the SAS System assumes that the data in the PUT statement is to be added as a new record. If you are processing a KSDS, you must specify a new primary key in the PUT statement. If you are processing an RRDS, you must specify an empty relative record slot with the RRN= option in the INFILE statement.

When you do have an INPUT statement that is associated with the INFILE statement (that is, you are reading from the data set before writing), the SAS System assumes that the data in the PUT statement is to update the record you have just read with the INPUT statement unless you have

- changed the RBA= variable value before the PUT statement executes for an ESDS. See “Combined Operations on an ESDS” on page 53 for more information.
- changed the primary key with PUT @ statements for a KSDS. The primary key is also changed if you do not copy it from the INPUT buffer because the key is blank in the PUT buffer. See “Combined Operations on a KSDS” on page 67 for more information.
- changed the RRN= variable value before the PUT statement executes for an RRDS. See “Combined Operations on an RRDS” on page 82 for more information.

Examples of Using VSAM Data in SAS Programs

The following examples illustrate the use of VSAM data in SAS procedures and programs. The examples are based on the STUDENT data set described in Appendix 2.

Generating PROC PRINT Listings from a KSDS

This example generates the following PRINT procedure listings:

1. all records read sequentially from the KSDS
2. the keys that are used to subset the KSDS
3. the subset of records read from the KSDS using keys from the second item in this list.

The example can generate PROC PRINT listings under CMS and OS/390.

```sas
/* This DATA step reads all of the records from a KSDS */
/* using sequential access. */
data test;
   infile myksds vsam;
   input id $9. lastname $10. frstname $10. address $25. city $15.
        state $2. zip $5. balance $5. gpa $4. class $2. hrs $2.
        finaid $1.;
run;
```
/* Generate a listing of all the records of a KSDS. */
proc print data=test;
run;

/* This DATA step subsets the keys to every third key. */
data keys;
  infile myksds vsam keypos=kpos keylen=klen;
  input @kpos key $varying200. klen;
  if mod(_n_,3)=0 then output;
/* This DATA step reads every third record from a KSDS using keyed access. */
data test;
  set keys;
  infile myksds vsam key=key;
  input id $9. lastname $10. frstname $10. address $25. city $15.
    state $2. zip $5. balance $5. gpa $4. class $2. hrs $2.
    finaid $1.;
run;

/* Generate a listing of the subset of keys. */
proc print data=keys;
/* Generate a listing of the subset of KSDS records. */
proc print data=test;
run;

---

**Generating Reports Using PROC MEANS**

This example reads all of the records from the KSDS that is described in the previous section, creates a numeric variable containing the students' GPA, and generates the following reports by using the MEANS procedure:

1. grade point average for all students
2. grade point average of students by class
3. grade point average of students by state.

The data must be sorted by the variable that is used in the BY statement of PROC MEANS before you run PROC MEANS. This example can also run under CMS and OS/390.

/* This DATA step reads all of the records from the KSDS and */
/* generates a numeric variable GPANUM from the character */
/* variable GPA using the input function. */
data test;
  infile myksds vsam;
  input id $9. lastname $10. frstname $10. address $25. city $15.
    state $2. zip $5. balance $5. gpa $4. class $2. hrs $2.
    finaid $1.;
  gpanum=input(gpa,4.2);
run;

proc means data=test;
  var gpanum;
Using a Windowing Program to Update VSAM Records

The following program is an example of using SAS Screen Control Language (SCL) to create a simple windowing application for updating VSAM records on OS/390.

This application uses SAS Screen Control Language to provide an interface to a VSAM KSDS. The data set is made up of student records, keyed by the student’s social security number (SSN). The application enables users to scroll through records using the NEXT and PREV buttons (or the forward and backward commands). The user can also retrieve a record with a specific key by entering a social security number and selecting the RETRIEVE button. Once a record is displayed, any desired changes can be made to the values in the window, and the record is updated by selecting the CHANGE button. A new record can be added by entering new values for all fields in the window and selecting the ADD button.

The application uses the following algorithm:

1. Create a SAS data set by reading records from the VSAM KSDS.
2. Display the values from the first record.
3. Perform the appropriate action when the user selects a button.
4. When the user chooses to quit the application,
   a. sort the SAS data set in ascending order by SSN
   b. delete the old VSAM KSDS
   c. create a new VSAM KSDS
   d. write all records from the SAS data set to new KSDS.

The purpose of this application is to illustrate how SCL can be used to create an interactive interface to the records in a VSAM data set. In the interest of clarity of the code, this application does little error checking.

INIT:

```sas
/* Set the VSAMLOAD and VSAMUPDATE SAS options. */
/* Assign a fileref to the VSAM data set. */
/* Read records into a SAS data set from a VSAM KSDS. */
/* Deallocate the fileref. */
/* Open the SAS data set for processing. */
```
control asis;
error=0;
submit continue STATUS;
option vsamload vsamupdate;
filename myksds 'dsname.ksds.student' disp=shr;

data stdrecs;
infile myksds vsam;
input id $9.
   lastname $10.
   firstname $10.
   address $25.
   city $15.
   state $2.
   zip $5.
   balance $5.
   gpa $4.
   class $2.
   hrs $2.
   finaid $1.;
run;

filename myksds clear;
endsubmit;
dsid=open('work.stdrecs','u');
prevrec=0;
nextrec=2;
rc=fetchobs(dsid,1);
link readval;
return;

MAIN:
/* Determine what the user wants to do, and perform */
/* the appropriate action. */
/* length cmd $ 10 idnum $ 9; */
length cmd $ 10;
cmd='';
put 'in MAIN - cmd = ' cmd;
call notify('RETRIEVE',_getText',cmd);
if (cmd = 'RETRIEVE') then do;
   put cmd=;
   link retrieve;
   return;
end;
call notify('CHANGE',_getText',cmd);
if (cmd = 'CHANGE') then do;
   put cmd=;
   link change;
   return;
end;
call notify('ADD',_getText',cmd);
if (cmd = 'ADD') then do;
   put cmd=;
   link add;
return;
end;
/* call notify('NEXT', '_getText', cmd); */
if (cmd = 'NEXT') then do;
    put cmd=;
    /* link next; */
    return;
end;
call notify('PREV', '_getText', cmd);
if (cmd = 'PREV') then do;
    put cmd=;
    link prev;
    return;
end;
call notify('QUIT', '_getText', cmd);
if (cmd = 'QUIT') then do;
    put cmd=;
    goto term;
    return;
end;
    cmd=''
return;
TERM:

/* Close the SAS data set. Sort the SAS data set by the */
/* variable holding the VSAM key value. Delete the old */
/* VSAM data set. Create a new VSAM data set to hold */
/* the updated records. (Note: The method used here */
/* (building a SAS macro to be submitted to the operating */
/* system command processor) only works on the OS/390 operating */
/* system.) Assign a fileref to the newly created VSAM data */
/* set. Write the records from the SAS data set to the VSAM */
/* data set. Deallocate the fileref. */
call close(dsid);
submit terminate;
proc sort data=work.stdrecs;
    by id;
run;

x "delete ('dsname.ksds.student') purge cluster";

%let mac=%str(define cluster %(name('dsname.ksds.student') ) );
%let mac=%mac %str(records(10 5) );
%let mac=%mac %str(recsz(90 90) );
%let mac=%mac %str(shareoptions(2,3) );
%let mac=%mac %str(reuse );
%let mac=%mac %str(volumes(APP004) );
%let mac=%mac %str(cisz(2048) );
%let mac=%mac %str(keys(9 0)%);  
%let mac=%mac %str(data );
%let mac=%mac %str($%(name('dsname.ksds.student.data') ) );
%let mac=%mac %str(cisz(2048) );
%let mac=%mac %str(index );
%let mac=&mac %str(%(name('dsname.ksds.student.index') )
%let mac=&mac %str(cisz(512)%

/* Submit the macro variable for execution. */
%sysexec &mac;

filename myksds ‘dsname.ksds.student’ disp=shr;

data _null_;
  set work.stdrecs;
  file myksds vsam reset;

/* Write the data from the variables in the SAS data set to */
/* the appropriate column in the current record of the KSDS. */
if id ^= ‘ ’ then do;
  put @1 id $9. /* Student’s social security number */
    @10 lastname $10. /* Student’s surname */
    @20 firstname $10. /* Student’s given name */
    @30 address $25. /* Permanent mailing address */
    @55 city $15. /* City of residence */
    @70 state $2. /* State of residence */
    @72 zip $5. /* Five-digit ZIP code */
    @77 balance $5. /* Balance from previous semester */
    @82 gpa $4. /* Grade point average on a4.00 scale */
    @86 class $2. /* FR, SO, JU, SE, or GR */
    @88 hrs $2. /* Hours registered for in next semester */
    @90 finaid $1. /* Financial aid eligibility, Y or N */
  end;
run;
filename myksds clear;
endsubmit;
return;

RETRIEVE:
/* Use a WHERE clause to subset the data set to contain only */
/* the record associated with the requested ID number. If */
/* there is an observation left in the data set, display its */
/* values. If there are no observations left in the data set, */
/* blank out any values in fields other than idnum and notify */
/* the user that there was no match found. */
clause="id=''||idnum|'||'";
rc=where(dsid,clause); rc=fetchobs(dsid,1);
if rc=0 then
  link readval;
else do;
  link blanks;
    _msg_='No matching record found.';
  end;
return;

CHANGE:
/* Update the values in the current observation. */
link writeval;
if error then
  error=0;
else
  rc=update(dsid);
return;

ADD:
/* Check to see if a record with that SSN already exists. If */
/* so, notify the user. Else add a new observation to the */
/* data set and update its variables. */
clause="id='"||idnum||"'";
prompt clause=;
rc=where(dsid,clause);
prompt rc=;
rc=fetchobs(dsid,1);
prompt rc=;
if rc=0 then do;
  _msg_='A record with that key already exists.';
  _msg_='No duplicates allowed';
end;
else do;
  rc=append(dsid);
  link writeval;
end;
if error then
  error=0;
else
  rc=update(dsid);
return;

NEXT:
  put 'next - nextrec = ' nextrec;
  put 'next - prevrec = ' prevrec;
  rc=fetchobs(dsid,nextrec);
  put rc=;
if rc=0 then do;
  prevrec=prevrec+1;
  nextrec=nextrec+1;
  link readval;
end;
else
  _msg_='NOTE: At bottom.';
return;

PREV:
  put 'prev - nextrec = ' nextrec;
  put 'prev - prevrec = ' prevrec;
if prevrec>0 then do;
  rc=fetchobs(dsid,prevrec);
  put rc=;
if rc=0 then do;
  prevrec=prevrec-1;
nextrec=nextrec-1;
   link readval;
end;
else
   msg_="NOTE: At top."
end;
else
   msg_="NOTE: At top."
return;

BLANKS:

   /* Blank out all values on the screen. */
   idnum =""
   lname =""
   fname =""
   address= ""
   city =""
   s ="
   zip ="
   bal ="
   gpa ="
   c ="
   h ="
   fa ="
return;

READVAL:
   /* Assign the screen variables the values contained in the */
   /* current observation. */
   idnum =getvarc(dsid,1);
   lname =getvarc(dsid,2);
   fname =getvarc(dsid,3);
   address= getvarc(dsid,4);
   city =getvarc(dsid,5);
   s =getvarc(dsid,6);
   zip =getvarc(dsid,7);
   bal =put(input(getvarc(dsid,8),5.),dollar10.2);
   gpa =getvarc(dsid,9);
   c =getvarc(dsid,10);
   h =getvarc(dsid,11);
   if getvarc(dsid,12)='Y' then
      fa='Yes'
   else
      fa='No'
return;

WRITEVAL:

   /* Write the values contained in the screen variables to the */
   /* variables in the current observation. */
   length tempbal $ 10;
call putvarc(dsid,1,idnum);
call putvarc(dsid,2,lname);
call putvarc(dsid,3,fname);
call putvarc(dsid,4,address);
call putvarc(dsid,5,city);
call putvarc(dsid,6,s);
call putvarc(dsid,7,zip);
tempbal=substr(bal,2);
pos=index(tempbal,'.');
if pos>0 then
    tempbal=substr(tempbal,1,pos-1)||substr(tempbal,pos+1);
tempbal=substr(tempbal,1,index(tempbal,'.')-1);
call putvarc(dsid,8,tempbal);
call putvarc(dsid,9,gpa);
call putvarc(dsid,10,c);
call putvarc(dsid,11,h);
temp=upcase(substr(fa,1,1));
if (temp='Y') | (temp='N') then
    call putvarc(dsid,12,temp);
else do;
    _msg_='Invalid value for Financial Aid Eligibility,(Yes or No)';
    error=1;
end;
return;

This example creates the window shown below.

Display 3.1  Result of SCL Windowing Program

To retrieve a record, enter the student's SSN and select RETRIEVE. To update a record, first RETRIEVE it, make changes and select CHANGE. To add a record, type in all information and select ADD. To quit, select QUIT.
CHAPTER 4

Defining and Loading the Sample VSAM Data Sets

Introduction

This chapter explains how to define and load a VSAM data set under the CMS and OS/390 operating systems.

Defining a VSAM Data Set under CMS

Before you can define and load a VSAM data set under CMS, you must first define and format a DOS disk and then define a master catalog on that disk.

Defining a VSAM Catalog

This example shows the commands that are needed to define the master VSAM catalog. If the minidisk has not been formatted, format it or a temporary disk with Device Support Facilities. Consult the appropriate IBM documentation for additional information on using Device Support Facilities.

The following commands create a VSAM master catalog. (Replace userid, virtual-address1, virtual-address2, and filemode with actual values from your site.)

LINK userid virtual-address1 virtual-address2 MR
ACCESS virtual-address2 filemode
DLBL IJSYSC B DSN NASTCAT (PERM EXTENT VSAM

After you enter the DLBL statement, the system prompts you for extent information with the following message:

DMSDLB331R ENTER EXTENT SPECIFICATIONS:
Your response to the system should be of the form

```
rrr nnn
```

In this case, `rrr` is the relative track and `nnn` is the number of tracks. Follow this response with a blank line.

Invoke the XEDIT editor to create a data set that defines the catalog. The data set should have filetype AMSERV as follows:

```
XEDIT MASTCAT AMSERV A
```

The AMSERV data set contains the following DEFINE command:

```
DEFINE MASTERCATALOG -
  (NAME (MASTCAT) VOLUME(CMSXXX) -
  CYL(X) FILE(IJSYSCT))
```

Exit XEDIT and issue the following AMSERV command to create the catalog:

```
AMSERV MASTCAT
```

---

**Defining an ESDS**

This example shows the commands that are needed to define an ESDS under CMS. The data set is the STUDENT data set that is used in Chapter 5, “Processing an ESDS in a SAS Job,” on page 47. For each VSAM data set that you want to define, you must create a definition file of type AMSERV, assign logical catalog and data set names, and then execute the AMSERV file.

To create the definition file, invoke XEDIT:

```
XEDIT VSAMTEST AMSERV A
```

Include the DELETE and DEFINE commands in the definition file:

```
DELETE -
  dsname.ESDS.STUDENT -
PURGE -
CLUSTER

/* This job will send back a return code of 8 during its */
/* first execution because the deleted VSAM data set does */
/* not yet exist. The job continues execution. */
DEFINE CLUSTER (NAME(dsname.ESDS.STUDENT) -
  RECORDS(10 5) -
  RECSZ(90 90)-
  SHAREOPTIONS(2,3) -
  REUSE -
  NONINDEXED -
  VOLUMES(xxxx) -
  CISZ(2048)) -
  DATA (NAME(dsname.ESDS.STUDENT.DATA)) -
  CATALOG (mycat) -
```

KSDS and RRDS are similarly defined, except that for a KSDS, you would add KEY and INDEX statements, and for an RRDS, you would add the NUMBERED statement.

Issue the following commands to assign logical names (again, replace `userid`, `virtual-address1`, `virtual-address2`, `filemode` and other italicized variables with actual values from your site):

```
Defining a VSAM Data Set under OS/390

On OS/390, you define a VSAM data set by using the IBM Access Method Services (AMS) IDCAMS utility, which is invoked from JCL. This example uses IDCAMS to delete, allocate, and define a KSDS, an RRDS, and an ESDS. Note that the IDCAMS DEFINE parameters are generally self-explanatory by name. Make special note of the parameters RECORDSIZE (average maximum) and KEYS (length offset), where the keys offset is relative to the beginning of the record.

```
//VSAMDEF JOB job information
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=A
//SYSIN DD *

DELETE (dsname.K1719) PURGE CLUSTER
DEFINE CLUSTER (NAME(dsname.K1719) INDEXED VOLUMES(xxxxxx) -
    TRACKS(1) KEYS(17 19) RECORDSIZE(40 110) NONSPANNED) -
    DATA (NAME(dsname.K1719.DATA)) INDEX (NAME(dsname.K1719.INDEX))

DELETE (dsname.R002) PURGE CLUSTER
DEFINE CLUSTER (NAME(dsname.R002) NUMBERED VOLUMES(xxxxxx) TRACKS(1) -
    RECORDSIZE(120 120) NONSPANNED) DATA (NAME(dsname.R002.DATA))

DELETE (dsname.E002) PURGE CLUSTER
DEFINE CLUSTER (NAME(dsname.E002) NONINDEXED VOLUMES(xxxxxx) -
    TRACKS(1) RECORDSIZE(80 80)) DATA (NAME(dsname.E002.DATA))

/*
*/
```

If the VSAM data sets do not already exist, this example produces a return code of 8 for the DELETE operation.

On OS/390, you can also define a VSAM data set in two other ways: by building a SAS macro variable, and by issuing a TSO DEFINE command in the Program Editor.
with the X statement. The SAS system options VSAMLOAD and VSAMUPDATE are necessary for loading and updating VSAM data sets.

The following is an example of a macro variable:

```sas
options vsamload vsamupdate;
/* Delete the cluster if it exists. */
x "delete ('dsname.esds.student') purge cluster;"

/* Build a macro variable containing the commands */
/* that define a VSAM ESDS. */
%let def=%str(define cluster %(name('dsname.esds.student') );
%let def=&def %str(records(10 5) );
%let def=&def %str(recsz(90 90) );
%let def=&def %str(shareoptions(2,3) );
%let def=&def %str(volumes( xxxxxx ) );
%let def=&def %str(reuse );
%let def=&def %str(cisz(2048) );
%let def=&def %str(nonindexed %) );

/* Submit the macro variable for execution. */
%sysexec &def;
run;
```

The example defines an ESDS that is named `dsname.ESDS.STUDENT`. If the ESDS already exists, this example deletes the data set and redefines it. The necessary SAS system options VSAMLOAD and VSAMUPDATE are included in the beginning of the example. The first qualifier of the data set name, `dsname`, represents a value that the user supplies.

The following is an example of a TSO DEFINE command:

```sas
X DEFINE CLUSTER
(  
    NAME('dsname.TEST.VSAMFILE.CLUSTER')
    VOLUME(xxxxxx)
    TRACKS(5,1)
    CONTROLINTERVALSIZE(4096)
    FREESPACEx(10,20)
    KEYS(4,0)
    RECORDSIZE(80,80)
  )
DATA
(  
    NAME('dsname.TEST.VSAMFILE.DATA')
  )
INDEX
(  
    NAME('dsname.TEST.VSAMFILE.INDEX')
    CONTROLINTERVALSIZE(1024)
  )
);
```

This example defines a VSAM data set named `dsname.TEST.VSAMFILE`. 
Loading Records into a VSAM Data Set

The following sections discuss loading records into VSAM data sets. These sections describe the options that you can use when you load records, the access types, and examples of loading and reloading records into VSAM data sets.

Loading Records into a New VSAM Data Set

VSAM does not allow you to process records while you are loading the data set. You can put the initial records into the data set, but you cannot read, update, or erase any of these records until the data set is closed. Because of this restriction, the SAS System requires you to use only a FILE statement, instead of both an INFILE and a FILE statement, for a VSAM data set that is to be loaded. You must also specify the SAS system option VSAMLOAD. Once the data set is loaded and closed, you can add, update, or erase records when the SAS system option VSAMUPDATE is in effect.

You can read from and write to records in any other data set within the same DATA step that you use to load a VSAM data set. For example, you can load a VSAM data set based on records you are processing from another data set.

VSAM requires you to load a KSDS sequentially in key order. You can load an RRDS either sequentially in record order or directly by using the RRN= direct-access option in the FILE statement. An ESDS can only be loaded sequentially.

Options Used When Loading

When you load initial records into a new VSAM data set, use only a FILE statement and specify the SAS system option VSAMLOAD. In addition, you can use the following options in the FILE statement when you load a VSAM data set:

- BUFND=
- BUFNI=
- COL=
- FEEDBACK=
- KEYLEN=
- KEYPOS=
- LINE=
- LINESIZE=
- N=
- PASSWD=
- RECORDS=
- RC4STOP
- RESET
- RRN=
- VSAM

Access Types When Loading

When you load records into a VSAM data set, access depends on the data set type:

- For an ESDS, load access is sequential; that is, the records are loaded in the order in which you write them. You cannot change this order after loading the data set.
- For a KSDS, load access is sequential; that is, you must load the records in key order.

This VSAM restriction is imposed for performance reasons. If you attempt to load a record with a key lower than that of a previous record, VSAM returns a logical error with a feedback code of 12. If the sequential load restriction is a problem, load one or more records into the data set, and then access the data set a second time in another DATA step. Once the data set is closed with one or more records, the load restrictions no longer apply. Loading the data set in this manner may take more computer resources than loading the data set sequentially. If the data to be loaded is in a SAS data set, you can sort it in primary key order with the SORT procedure before loading the VSAM data set.
For an RRDS, load access is sequential unless you specify the RRN= direct-access variable in the FILE statement. The first record is loaded into the first slot, and each subsequent record is loaded into the next successive slot unless you explicitly load the records in some other order by using the RRN= variable.

---

**Reloading a VSAM Data Set**

If you plan to reload data sets into an existing VSAM data set in a DATA step, keep the following points in mind:

- You must define the data set with the VSAM option REUSE.
- You must specify the SAS system option VSAMLOAD.
- Use the RESET option in the FILE statement to reset the existing data set to empty (no records) when it is opened. If the VSAM data set is not defined with the REUSE option and you attempt to use the RESET option, the DATA step will not execute because VSAM does not open the data set.
- Reload the empty data set with new records.

Data sets that have alternate indexes cannot be reloaded; they must be deleted, defined, and then loaded.

---

**Loading a VSAM Data Set in a SAS DATA Step**

The following example shows how to load a VSAM data set in a SAS DATA step. The data set is described in Appendix 2, “Sample STUDENT Data Set,” on page 107.

In the example, a previously defined ESDS is loaded in a SAS DATA step. The example also applies to a KSDS and an RRDS.

```sas
data load;
  /* Open a SAS data set for input. */
  set vsamdata.student;

  /* Open previously defined VSAM ESDS for output. */
  file myesds vsam;

  /* Write the data from the variable in the SAS data set to */
  /* the appropriate column in a record of the ESDS. */
  put @1 id $9. /* Student’s Social Security number */
  @10 lastname $10. /* Student’s surname */
  @20 firstname $10. /* Student’s given name */
  @30 address $25. /* Permanent mailing address */
  @55 city $15. /* City of residence */
  @70 state $2. /* State of residence */
  @72 zip $5. /* Five-digit ZIP code */
  @77 balance $5. /* Balance from previous semester (if any) */
  @82 gpa $4. /* Grade point average on a 4.00 scale */
  @86 class $2. /* FR, SO, JU, SE, or, GR */
  @88 hrs $2. /* Hours registered for in next semester */
  @90 finaid $1. /* Financial aid eligibility, Y or N */
run;
```
INTRODUCTION

In Chapter 1 you were introduced to three types of data set organization in VSAM. They are
- ESDS
- KSDS
- RRDS.

This chapter deals extensively with processing an ESDS in SAS jobs. You may recall from Chapter 1 that record storage in an ESDS is determined by the order in which the records are entered into the data set without respect to the record contents, and that new records are stored at the end of the data set. Also, remember that an ESDS is appropriate for applications that do not require any particular ordering of the data by the record contents or for those that require time-ordered data. Applications that use a log or journal are well suited for an ESDS data set structure.

This chapter explains how you can read, add, and update ESDS records in SAS programs. The options associated with each of these operations are described. In many cases, the option's meaning depends on how it is used within a SAS program. (Loading ESDS records is discussed separately in Chapter 4, “Defining and Loading the Sample VSAM Data Sets,” on page 41.)

This chapter presents three helpful tables: Table 5.1 on page 48 lists the options that are significant for processing ESDS records, Table 5.2 on page 49 summarizes the access type that you can use for each ESDS operation, and Table 5.3 on page 50 points out the type of access that you can use for an ESDS with an alternate index. Finally, there are examples of reading, adding, updating, and performing combined operations on an ESDS data set. The examples are based on the STUDENT data set defined and loaded in Chapter 4 and described in Appendix 2, “Sample STUDENT Data Set,” on page 107.
For information on using a KSDS in SAS jobs, refer to Chapter 6, “Processing a KSDS in a SAS Job,” on page 55. For information on using an RRDS in SAS jobs, refer to Chapter 7, “Processing an RRDS in a SAS Job,” on page 73.

---

### Special SAS Options Used with an ESDS

The special SAS options fall into functional categories. The following table lists the special SAS options and categories used for processing an ESDS data set. Informational and record retrieval options are specified in the INFILE statement. Record storage options are specified in the FILE statement.

**Table 5.1  SAS Options for an ESDS**

<table>
<thead>
<tr>
<th>Option</th>
<th>Informational</th>
<th>Record Retrieval</th>
<th>Record Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKWARD</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BUFND=</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BUFNI=</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONTROLINTERVAL</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ERRORABEND</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FEEDBACK=</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PASSWD=</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RBA=</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RC4STOP</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECORDS=</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SEQUENTIAL</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>UPDATE=</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>VSAM</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

---

### Reading Records from an ESDS

You can use sequential or addressed direct access to read records from an ESDS within a SAS program (see Table 5.2 on page 49). If the ESDS has an alternate key index, you can also use keyed direct access. See “Keyed Direct Access by Alternate Keys” on page 50 for details. The options that are specified in the INFILE statement determine the access type for an ESDS read operation. Either the VSAMREAD or the VSAMUPDATE global SAS system option must be specified in order to read VSAM data sets.
Table 5.2 Access Types for ESDS Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Read (INFILE/INPUT statements)</th>
<th>Write (FILE/PUT statements)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Sequential Direct with RBA= option</td>
<td>Does not apply</td>
</tr>
<tr>
<td>Add*</td>
<td>Sequential with RBA= and SEQ options Direct with RBA= option</td>
<td>Sequential: records are always added to the end of file</td>
</tr>
<tr>
<td>Update</td>
<td>Sequential Direct with RBA= option</td>
<td>Direct: the last record read is the record updated</td>
</tr>
<tr>
<td>Load</td>
<td>Does not apply</td>
<td>Sequential: in entry order</td>
</tr>
</tbody>
</table>

* The INPUT statement is not required.

Reading an ESDS with Sequential Access

In an ESDS, sequential means in entry order. By default, the records that are in the data set are read from the beginning to the end of the data set. The following example shows the DATA step that you can use to read an ESDS sequentially:

```sas
/* Read data from an ESDS into a SAS data set */

data one;
  infile myesds vsam;
  input;
  ...more SAS statements...

If you specify the BACKWARD option, the data set is read backward, from the last record to the first record.

Reading an ESDS with Direct Access

You can read ESDS records with two kinds of direct access: addressed (by RBA) and keyed (by alternate key index).

Addressed Direct Access by RBA

When an ESDS is read with addressed direct access, records are retrieved directly by the address relative to the beginning of the data set (relative byte address). For this type of access to be useful, you must know the RBAs of the records that you want to access. You may know the RBA if it has some relationship to the record contents or if you have obtained it, for example, from the _RBA_ automatic variable in a previous SAS DATA step.

To use addressed direct access, specify the RBA= option in the INFILE statement for the VSAM data set that is to be accessed by RBA. The RBA= option defines a variable whose value must be set to the RBA of the logical record or control interval to be retrieved by an INPUT statement. The address you specify must correspond to the
beginning of a data record (logical record or control interval). Otherwise, the request is invalid and causes a VSAM logical error.

The following sample program illustrates addressed direct access to an ESDS:

```sas
data one;
   infile myesds vsam;
   input;
   rbanum=_rba_;  
   keep rbanum;
run;

data two;
   set one;
   infile myesds vsam rba=rbanum;
   input;
   ...more SAS statements...
```

**Keyed Direct Access by Alternate Keys**

If there is an alternate key index for an ESDS, you can use keyed direct access by alternate keys to read an ESDS. An alternate index is created outside the SAS environment by using IBM Access Method Services. Then the ESDS records can be accessed directly by key, in a manner similar to a KSDS. (For an introduction to the alternate index concept, refer to “Keyed direct access with an alternate index” on page 12. For instructions on how to create an alternate index for an ESDS, see Chapter 8, “Using Alternate Indexes for VSAM Data Sets,” on page 87.

You can treat an ESDS accessed through an alternate index as if it were a KSDS, except that

- records cannot be erased
- the record length cannot be changed.

The following table summarizes the type of access you can use for an ESDS with an alternate key index.

**Table 5.3  Operations on an ESDS with an Alternate Index**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Sequential</td>
</tr>
<tr>
<td></td>
<td>Direct by:</td>
</tr>
<tr>
<td></td>
<td>alternate key</td>
</tr>
<tr>
<td></td>
<td>RBA</td>
</tr>
<tr>
<td>Add</td>
<td>Sequential</td>
</tr>
</tbody>
</table>
Adding Records to an ESDS

To add records to an existing ESDS:

1. specify the SAS system option VSAMUPDATE.

2. include both an INFILE and a FILE statement for the data set. Specify the VSAM option in both the INFILE and the FILE statements. Specify all other options in the INFILE statement, which must precede the FILE statement.

For a list of the options that you can use to add records to an ESDS, refer to Table 5.1 on page 48.

You do not have to include an INPUT statement with the INFILE statement. The INPUT statement is unnecessary because you do not have to read a record in order to add a new record. For example,

```
data three;
  infile myesds vsam ;
  file myesds vsam ;
id='289478363';
lastname='Cox ';
firstname='June ';
address='Rt. 2 Box 784 ';
city='Cheyenne';
state='WY';
zip='59334 ';
balance='00100';
gpa='2.33';
class='SE';
hrs='13';
finaid='Y';
if _n_=1 then newstu=0;
  put @1 id $9. /* Student’s Social Security number */
     @10 lastname $10. /* Student’s surname */
     @20 firstname $10. /* Student’s given name */
     @30 address $25. /* Permanent mailing address */
     @65 city $15. /* City of residence */
     @70 state $2. /* State of residence */
     @72 zip $5. /* Five-digit ZIP code */
     @77 balance $5. /* Balance from previous semester (if any) */
     @82 gpa $4. /* Grade point average on a 4.00 scale */
     @86 class $2. /* FR, SO, JU, SE, or GR */
     @88 hrs $2. /* Hours registered for in next semester */
     @90 finaid $1. /* Financial aid eligibility, Y or N */
   newstu=newstu+1;
```
Updating Records in an ESDS

To update records in an ESDS:

1. specify the SAS system option VSAMUPDATE.
2. include both an INFILE and a FILE statement for the data set. Specify the VSAM option in both the INFILE and the FILE statements. Specify all other necessary options in the INFILE statement, which must precede the FILE statement.
3. use an INPUT statement to read the record being modified.
4. use the PUT statement to write the complete record. An INPUT statement brings the record into the INPUT buffer but does not copy it to the PUT buffer. This enables you to change the record easily.

For a list of options that you can use to update records in an ESDS, refer to Table 5.1 on page 48.

When records in an ESDS are updated

- input access for reading can be either sequential or direct. Access is sequential unless you specify the RBA= direct access option in the INFILE statement (see Table 5.2 on page 49).
- output access for writing is direct because the last record that is read is the record updated.

Using the PUT Statement When Updating

When you update an ESDS record, you must use the PUT statement to write the complete record. There are two common ways of writing the record with the PUT statement:

- Build the complete record by specifying all fields with the PUT statement. This method might be best when many of the fields need to be updated.
- Copy the input record to the output buffer (with PUT _INFILE_) and overlay selected fields. This method might be best when relatively few fields need to be updated.

The latter method is the easiest for most applications. For example, the following statement copies the last record that is read into the PUT buffer and overlays the information starting in columns 10 and 30 with the values in NEWDATA1 and NEWDATA2:

```sas
PUT @ 1 _INFILE_
  @ 10 NEWDATA1
  @ 30 NEWDATA2;
```

In most cases, when a record is retrieved for update, no user, including you, can access that particular record or any other records in the same control interval. Use the
UPDATE= option to avoid user lockout when only a few of the records retrieved need to be updated. See “Using the UPDATE= Option” on page 31 for more information.

In the example below, the RBAs of records are captured and stored in a SAS variable called RBAVAR from the _RBA_ automatic variable. In the next DATA step the records are then read without being retrieved for update until the condition specified in the IF clause is met. When the IF condition is true (RBANUM=1260), the record is retrieved again with update access.

```
data rbas;
  infile myesds vsam ;
  input;
  rbanum=_rba_ ;
  keep rbanum;
run;

data esdsupdt;
  set rbas;
  updtevar=0;
  infile myesds vsam rba=rbanum update=updtevar;
  input;
  if (rbanum=1260) then do;
    updtevar=1;
    input;
    /* Create NEWDATA */
    lastname='Flintstone ';
    frstname='Fred '
    file myesds vsam ;
    put @1 _infile_
      @10 lastname
      @20 firstname;
  end;
run;
```

Combined Operations on an ESDS

You may want to perform more than one operation on an ESDS in one DATA step. For example, perhaps you want to read some records, update other records, and add new records in one DATA step. Regardless of the number of different operations, you need to specify only one pair of INFILE and FILE statements for the entire DATA step. Specify the VSAM option in both the INFILE and the FILE statements. Specify all other options that you may need to process the ESDS in its INFILE statement.

In a DATA step that combines operations, the SAS System determines whether you want to add new records or update existing ESDS records based on whether an INPUT statement is associated with the INFILE statement.

- When you do not have an INPUT statement associated with the INFILE statement, the SAS System assumes that the data in the PUT statement is to be added as a new record at the end of the data set.
- When you do have an INPUT statement associated with the INFILE statement, the SAS System assumes that the data in the PUT statement is to modify the record you have just read, unless you change the RBA= variable to a different value before the PUT statement executes. When you change the RBA= variable, whatever is in the PUT buffer is added as a new record to the end of the data set.
Adding Records after Reading

To add a new record after reading an existing record, set the RBA= variable to a different value before you execute the PUT statement. The new RBA value instructs VSAM not to update the last record retrieved with an INPUT statement; instead, it adds the data as a new record. (The actual value in the RBA= variable is ignored because VSAM chooses the RBA for a new record.) For example,

```plaintext
data four;
  set rbas;
  infile myesds vsam rba=rbanum;
  file myesds vsam;
  input;
  if (rbanum= 1080) then do;
    rbanum= 803;
    lastname='Rubble ';
    frstname='Barney ';
    file myesds vsam ;
    put @1 _infile_  
      @10 lastname  
      @20 frstname;
  end;
run;
```

In the example, MYESDS is read until RBANUM 1080 is found; then a record is added after 1080 because changing the RBANUM cancels the update.

If you want to read an ESDS sequentially while adding new records, specify the SEQUENTIAL option and the RBA= option in the INFILE statement. (The SEQUENTIAL option specifies sequential record retrieval when the RBA= direct access option indicates direct record storage for the PUT statement.)
CHAPTER 6

Processing a KSDS in a SAS Job

Introduction

In Chapter 1 you were introduced to three types of data set organization in VSAM:
- KSDS
- RRDS
- ESDS.

This chapter deals extensively with processing a KSDS in SAS jobs. You might recall from Chapter 1 that each record in a KSDS has a key that contains a unique value and that KSDS records are retrieved by their key sequences. The key is a contiguous portion of the record and is defined when the data set is created. Also, remember that a KSDS is always defined with a prime index that relates the record's key value to its relative location in the data set. VSAM uses the index to locate a record for retrieval and to locate a collating position for record insertion. A KSDS is the most flexible...
approach for most applications because the record can be accessed directly via the key field. Access is not dependent on the physical location of the record in the data set.

This chapter explains how you can read, add, update, and erase KSDS records in SAS programs. The options that are associated with each of these operations are described. In many cases, the option’s meaning depends on how it is used within a SAS program. (Loading KSDS records is discussed separately in Chapter 4, “Defining and Loading the Sample VSAM Data Sets,” on page 41.)

This chapter presents two helpful tables. Table 6.1 on page 56 lists the options that are significant for processing KSDS records and Table 6.2 on page 58 summarizes the access type for KSDS operations. Finally, there are examples of reading, adding, updating, and performing combined operations on a KSDS data set. The examples are based on the STUDENT data set described in Appendix 2, “Sample STUDENT Data Set,” on page 107. You can run the examples by using the sample programs provided in the online help system and following the steps to define and load a KSDS described in Chapter 4, “Defining and Loading the Sample VSAM Data Sets,” on page 41.

For information on using an ESDS in SAS jobs, refer to Chapter 5, “Processing an ESDS in a SAS Job,” on page 47. For information on using an RRDS, refer to Chapter 7, “Processing an RRDS in a SAS Job,” on page 73.

---

**Special SAS Options Used with a KSDS**

The special SAS options fall into functional categories. The following table lists the special SAS options and categories that are used for processing a KSDS data set. Informational and record retrieval options are specified in the INFILE statement. Record storage options are specified in the FILE statement.

### Table 6.1 SAS Options for a KSDS

<table>
<thead>
<tr>
<th>Option</th>
<th>Functional Category</th>
<th>Informational</th>
<th>Record Retrieval</th>
<th>Record Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKWARD</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BUFND=</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BUFNI=</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONTROLINTERVAL</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERASE=</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ERRORABEND</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FEEDBACK=</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GENKEY</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEY</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>KEYGE</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEYLEN=</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEYPOS=</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PASSWD=</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RBA=</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RC4STOP</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Reading Records from a KSDS**

You can read KSDS records with sequential access, direct access, and a combination of both sequential and direct access (see Table 6.2 on page 58). The type of KSDS read operation is specified with appropriate options in the SAS INFILE statement. Also, you must specify either the VSAMREAD or the VSAMUPDATE global SAS system option in order to read VSAM data sets.

**Reading a KSDS with Sequential Access**

By default, KSDS records are read in key order with sequential access; that is, they are read from the beginning to the end of the collating sequence of the key field contents. The following example shows the DATA step that you can use to read a KSDS sequentially:

```sas
data one;
  infile myksds vsam ;
  input;
  ...more SAS statements...
```

If you specify the BACKWARD option, the data set is read backward, from the highest key to the lowest.

**Reading a KSDS with Direct Access**

A KSDS is read directly by
- keyed direct access by key, approximate key, or generic key
- addressed direct access by RBA
- skip sequential access, which is a combination of direct and sequential access.

You cannot use both keyed direct and addressed direct access for the same data set in one DATA step.

**Keyed Direct Access to a KSDS**

To read a KSDS with keyed direct access, specify the key of the record you want the SAS System to read. The key can be one of the following:
- the exact key of the record
- an approximate key that is less than or equal to the actual key of the record
- a generic key specifying a leading portion of the key contained in records wanted.
Several of the INFILE statement options that are described in Chapter 2, “SAS Options and Variables for VSAM Processing,” on page 15 are used to retrieve KSDS records. These options are GENKEY, KEY=, KEYGE, KEYLEN=, and KEYPOS=.

Table 6.2 Access Types for KSDS Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Read (INFILE/INPUT Statements)</th>
<th>Write (FILE/PUT Statements)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Sequential</td>
<td>Does not apply</td>
</tr>
<tr>
<td></td>
<td>Direct by:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>□ key with KEY= option</td>
<td></td>
</tr>
<tr>
<td></td>
<td>□ generic key with GENKEY and KEYLEN= options</td>
<td></td>
</tr>
<tr>
<td></td>
<td>□ alternate key</td>
<td></td>
</tr>
<tr>
<td></td>
<td>□ RBA with RBA= option</td>
<td></td>
</tr>
<tr>
<td>Skip</td>
<td>sequential with SKIP and KEY= options</td>
<td></td>
</tr>
<tr>
<td>Add*</td>
<td>Sequential</td>
<td>Direct: specify a unique key in the PUT statement</td>
</tr>
<tr>
<td></td>
<td>Direct with KEY= option</td>
<td></td>
</tr>
<tr>
<td>Update</td>
<td>Sequential</td>
<td>Direct: prime key in the PUT statement must match the key of record read to update</td>
</tr>
<tr>
<td></td>
<td>Direct with:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>□ KEY= option</td>
<td></td>
</tr>
<tr>
<td></td>
<td>□ RBA= option</td>
<td></td>
</tr>
<tr>
<td>Erase</td>
<td>Sequential</td>
<td>Direct: the record that is read is the record that is erased</td>
</tr>
<tr>
<td></td>
<td>Direct with:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>□ KEY= option</td>
<td></td>
</tr>
<tr>
<td></td>
<td>□ RBA= option</td>
<td></td>
</tr>
<tr>
<td>Load</td>
<td>Does not apply</td>
<td>Sequential: in prime key order</td>
</tr>
</tbody>
</table>

* The INPUT statement is not required.

**KEY= option**

The direct access option KEY= defines a SAS variable whose value is the key of the record that you want to read with an INPUT statement. The following is a simple example of the use of the KEY= option:

```sas
data two;
    id= ’293652329’;
    keyvar= id;
    infile myksds vsam key=keyvar;
    input;
    ...more SAS statements...
```

In the example, VSAM retrieves the record with the ID value of 293652329 from the MYKSDS data set.
The KEY= option can specify a list of variables to create a key up to 256 characters in length. The key that is passed to VSAM is constructed by concatenating the variables specified; blanks are not trimmed.

Unless it is used with GENKEY, the key value that is passed to VSAM is either padded with blanks or truncated, as necessary, to equal the key length that is defined when the KSDS was created. For example, if the KSDS that is defined in Chapter 4 specified a key length of 5 instead of 9 characters, the key that is in the preceding example would be truncated to 29365 and only records that match that value would be retrieved. With the GENKEY option, SAS programs treat the value of the KEY= variable as a partial key so that length is not an issue.

**KEYGE option**

You can use the KEYGE option to specify that the read retrieval is to be any record whose key is equal to or greater than the key specified by the KEY= variable. This approximate key retrieval is useful when the exact key is not known. The KEYGE option applies to all records read from the data set in that DATA step; that is, you cannot turn KEYGE on and off.

The following example retrieves the first record that either matches or is greater than the key given; in this case, it is 600000000:

```sas
data three;
  id='600000000';
  keyvar=id;
  infile myksds vsam key=keyvar keyge;
  input;
  ...more SAS statements...
```

If necessary, the value of KEYVAR will be padded with blanks or truncated to equal the key length that was defined when the KSDS was created.

**GENKEY option**

The GENKEY option specifies generic key processing. With the GENKEY option, SAS programs treat the value given by the KEY= variable as a partial key (the leading portion) of the record that is to be read. The SAS System reads only the first record that contains the matching partial key (unless you also specify skip sequential processing). Changing the value of the KEY= variable indicates another generic key retrieval request. The GENKEY option applies to all records read from the data set in that DATA step; that is, you cannot turn GENKEY on and off.

The following example retrieves the first record with a key matching the first part of the key specified by the KEY= variable, KEYVAR:

```sas
data four;
  id='578';
  keyvar=id;
  infile myksds vsam key=keyvar genkey;
  input;
  ...more SAS statements...
```

The record that is read is the first record with 578 in its ID.

When you specify both the GENKEY and the SKIP options, the SAS System retrieves the first record that contains the matching partial key and then reads the following records sequentially. Access is sequential after the first record until you change the value of the KEY= variable, which indicates another direct-access, generic-key retrieval request. See “Reading a KSDS with Skip Sequential Access” on page 62 for more information and an example of how to use both the GENKEY and SKIP options.
KEYLEN= option

Use the KEYLEN= option with the GENKEY option to change the generic key length from one request to the next. KEYLEN= defines a SAS variable that specifies the length of the key to be compared to the keys in the data set. The variable's value is the number of generic key characters passed to VSAM. If you specify GENKEY without the KEYLEN= option, the generic key length is the KEY= variable length (or the sum of the KEY= variable lengths, if a list is specified) that is defined in the KSDS. The following example retrieves the first record that matches the first character of KEYVAR's value, which is 5:

```
data five;
  id='578';
  keyvar=id;
  klvar=1;
  infile myksds vsam key=keyvar genkey keylen=klvar;
  input;
  ...more SAS statements...
```

The KEYLEN= option has another use. It can also give information about the key field length to the application program. Before the DATA step executes, the SAS System sets the variable that is specified by KEYLEN= to the actual (maximum) key length that is defined in the KSDS data set. This option allows KSDS keys to be read without knowing the key length in advance. Assign the initial value of the KEYLEN= variable to a different variable if you also intend to set the KEYLEN= variable for generic key processing or if you need to know and use the key-length value later in the DATA step. You may need to name the variable in a RETAIN statement if you need this initial value after the first execution of the DATA step. For example,

```
data six;
  id='578';
  keyvar=id;
  infile myksds vsam key=keyvar genkey keylen=klvar;
  retain lenkey;
  lenkey=klvar;
  put lenkey=;
  klvar=1;
  input;
  ...more SAS statements...
```

In the example, the first two statements assign the key value of the records that are wanted to KEYVAR, the RETAIN statement captures and stores the initial value of the KEYLEN= variable into the LENKEY variable for later use as KLVAR, and then KLVAR is set to 1 for generic processing.

KEYPOS= option

The KEYPOS= option specifies a numeric SAS variable that VSAM sets to the position of the key in KSDS records before the DATA step executes. The variable is set to the column number, not the offset, which is the column number minus 1. This option allows you to read KSDS keys without knowing their positions in advance. For example,

```
data seven;
  length keyvar $9;
  infile myksds vsam keypos=kpvar;
  retain kpvar;
  input @kpvar keyvar;
  ...more SAS statements...
```
In the example, VSAM retrieves each record of the KSDS and stores the record key position in variable KPVAR. The records’ key value is read from the input buffer into character variable KEYVAR using the key position value.

It is possible to read KSDS keys without knowing either the key position or length in advance by using the KEYLEN= and the KEYPOS= options together. The SAS variables that you specify with the KEYLEN= and KEYPOS= options should not be present in any SAS data set that is used as input to the DATA step. Use an INPUT statement of the following form, where KPVAR is the KEYPOS= variable, KLVAR is the variable specified by the KEYLEN= option, and KEYVAR is a variable that contains the key. This example reads keys whose lengths are less than or equal to 2000.

```
infile myksds vsam key=keyvar keypos=kpvar keylen=klvar;
retain kpvar klvar;
input @kpvar keyvar $vary2000. klvar ... 
```

### Packed Decimal Data and Key Variables

You can use packed decimal data (date and time values) in a key variable if you request it in the same internal format as the VSAM data set. For a variable key, use the PUT function to produce the key in character format. For example, the following code writes the value 293652329 to the character variable KEYVAR using the packed decimal format PD5:

```
data dsname;
  id=293652329;
  keyvar=put(id,pd5.);
infile myksds vsam key=keyvar;
...more SAS statements...
```

For a single, known key or the leading portion of the key, use a hexadecimal value in your request as follows:

```
data dsname;
  keyvar='5789'x;
infile myksds vsam key=keyvar keyge;
...more SAS statements...
```

### Keyed Direct Access by Alternate Index

If there is an alternate key index for a KSDS, you can use keyed direct access by alternate keys. The advantage of an alternate index is that you can effectively rearrange records in the data set instead of keeping copies organized in separate ways for different applications. Refer to “Keyed direct access with an alternate index” on page 12 for an introduction to the alternate index concept and a list of references for the topic.

The main difference between the prime key and the alternate key is that there can be many alternate keys, and they can be defined as nonunique. This means that an alternate key can point to more than one record in the base cluster. For example, if an alternate index by course number is defined over a STUDENT data set that is organized by student id, several students could have the same course number. Each alternate index entry would point to several prime key records in the base cluster.

See Chapter 8, “Using Alternate Indexes for VSAM Data Sets,” on page 87 for examples of the control language that defines an alternate index over a KSDS for OS/390.
**Addressed Direct Access by RBA**

A KSDS can be read with addressed direct access, which means that a record is retrieved directly by its address. A record's address is relative to the beginning of the data set (relative byte address or RBA).

To indicate addressed access to KSDS records, use the RBA= option in the INFILE statement to specify the RBA of the record that you want. The RBA= option defines a SAS variable that you set to the RBA of the logical record or control interval that is to be retrieved by an INPUT statement. The address that you specify must correspond to the beginning of a data record (logical record or control interval); otherwise, the request causes a VSAM logical error. The RBA= variable is not added to the output data set. For example,

```sas
data rbas;
  infile myksds vsam;
  input;
  rbanum=_RBA_;
  keep rbanum;
run;

data eight;
  set rbas;
  infile myksds vsam rba=rbanum;
  input;
  ...more SAS statements...
```

**Reading a KSDS with Skip Sequential Access**

With skip sequential access, the initial record of a series is located with keyed direct access. (VSAM does not permit skip sequential addressed access.) Once the first record is obtained, subsequent records are retrieved sequentially. Skip sequential processing improves performance since sequential retrieval requires less overhead and is faster than direct retrieval. Skip sequential access is also useful when you know the key of the first record that you want but do not know (or do not want to specify) the key of subsequent records.

Use the SKIP option in the INFILE statement to specify skip sequential processing. Retrieve the first record directly by specifying the key of the record that you want with the KEY= option in the INFILE statement. When you use the SKIP option, leaving the value of the KEY= variable unchanged turns off direct access and indicates that subsequent records are to be retrieved with sequential access. If you need to know the key of subsequent KSDS records, you can read it from the record itself because the key is part of the record.

The following sample program illustrates skip sequential retrieval and generic key processing. The program reads in the generic portion of the key, reads all of the records in the KSDS data set with that generic key, and then writes them on the SAS print file. Note that the SKIP option retrieves only the first record with a key matching the KEY= variable. You must supply statements to read additional records.

When processing skip sequentially, remember that you must end the DATA step explicitly with a SET or a STOP statement. In the example program below, end-of-file sets the feedback code to 4, and the IF RC=4 clause stops the DATA step. If there is no record with the generic key specified, the FEEDBACK= variable is set to 16, a message is printed, and the next observation is processed.

```sas
data keys;
  length keyvar keyword1 $1;
```
input keyvar $;
cards;
1
5
8;

data process;
set keys;
file print;
if _n_=1 then do;
   put 'The KSDS records selected by GENKEY and SKIP are: ';
   put;
end;

   /* Read all the records with the value of KEYVAR in the key. */
   /* Set KEY= variable for generic skip sequential processing. */
   infile myksds vsam key=keyvar genkey skip feedback=sasrc keypos=kp;
   input @;
   /* Stop if end-of-file. */
   if sasrc=4 | sasrc=16 then do;
      _error_=0;
      if sasrc=4 then stop;
   /* If there is no record with this generic key, print a */
   /* message to the SAS print file, and go on to the next */
   /* observation. */
   /* Retain the value of KEYVAR to compare the first word of the */
   /* key of records read with sequential access. Initialize the */
   /* value of KEYWORD1 to the KEYVAR value to start the loop. */
   input @ kp keyword1 $;
   /* Sequentially read while the first word of the key matches */
   /* the value of KEYVAR. Write the records to the SAS print */
   /* file. */
   do while (keyword1 eq keyvar);
      put _infile_;
   input @;
   /* Stop if end-of-file. */
   if sasrc=4 | sasrc=16 then do;
      _error_=0;
      if sasrc=4 then stop;
   /* If there is no record with this generic key, print a */
   /* message to the SAS print file, and go on to the next */
Adding Records to a KSDS

To add records to a KSDS:

1. specify the SAS system option VSAMUPDATE.
2. include both an INFILE and a FILE statement for the data set. Specify the VSAM option in both the INFILE and the FILE statements. Specify all other options in the INFILE statement, which must precede the FILE statement.
3. use the PUT statement to write the record.

For a list of the options that you can use when adding records, refer to “Special SAS Options Used with a KSDS” on page 56.

When you add records to an existing KSDS, you do not have to include an INPUT statement with the associated INFILE statement. An INPUT statement is unnecessary in this case because you do not have to read a record to add a new record. For example,

```sas
data ten;
  infile myksds vsam;
  file myksds vsam;
  id='963215683';
  lastname='Flintstone';
  firstname='Pebbles';
  address='1234 Quarry Rd';
  city='Boulder';
  state='CO';
  zip='12345';
  balance='00555';
  gpa='3.33';
  class='FR';
  hrs='13';
  finaid='Y';
  put @1 id $9.
    @10 lastname $10.
    @20 firstname $10.
    @30 address $15.
    @55 city $15.
    @70 state $2.
    @72 zip $5.
    @77 balance $5.
    @82 gpa $4.
    @86 class $2.
```

```sas
/* observation. */
else do;
  sasrc=0;
  put 'There is no record with this generic key: ' keyvar;
  return;
end;
end;
input @ kp keyword1 $;
end;
run;
```
In the example, a record for a new student, PEBBLES FLINTSTONE, is defined and added to data set MYKSDS. The new record is added in the data set in ascending key order (in this case, according to the value of the ID variable) because output access is always direct when a new record is added to a KSDS (see Table 6.2 on page 58). The key for the record must be part of the PUT statement data. **You must specify a unique prime key in the PUT statement data.** VSAM does not allow duplicate prime keys. Keys do not have to be in ascending order during the update process.

**Updating Records in a KSDS**

To update records in a KSDS:

1. specify the SAS system option VSAMUPDATE.
2. include both an INFILE and a FILE statement for the data set. Specify the VSAM option in both the INFILE and the FILE statements. Specify all other necessary options in the INFILE statement, which must precede the FILE statement.
3. use an INPUT statement to read the record being modified. You must first retrieve the record sequentially or by direct access, using either the KEY= or RBA= option, before you can update the data set.
4. use the PUT statement to write the complete record.

For a list of the options that you can use when updating records, refer to “Special SAS Options Used with a KSDS” on page 56.

There are two common ways of writing the record with the PUT statement:

- Build the complete record by specifying all fields with the PUT statement. This method may be best when many of the fields need updating or when the updated record is shorter than the existing record because it avoids the problem of trying to eliminate or blank out the unwanted fields.
- Copy the input record to the output buffer (with PUT _INFILE_), and overlay selected fields in the copy. This method may be best when relatively few fields need to be updated.

The latter method is the easiest for most applications. The following statement copies the last record that is read into the PUT buffer and overlays the information starting in columns 10 and 30 with the values in NEWDATA1 and NEWDATA2:

```
put @ 1 _infile_
    @ 10 newdata1
    @ 30 newdata2;
```

**Note:** If you change the key of the record that was most recently retrieved, then the modified record is added as a new record. There is a VSAM restriction that does not allow you to change the primary key of a KSDS record.

In the following example, the SAS data set RKEYS contains the replacement data for a series of records in the data set MYKSDS. DATA1, DATA2, and KEYDATA are variables in the SAS data set RKEYS, which contains the new data and the VSAM key for records that are to be replaced.

```
data _null_
    set rkeys;
    infile mykeds vsam keypos=KP;
```

Erasing Records from a KSDS

To erase a record from a KSDS:

1. specify the SAS system option VSAMUPDATE.

2. use an INFILE statement and an INPUT statement to read the record and a FILE statement and a PUT statement to erase the record. Of course, the INFILE statement and FILE statement must have the same fileref; they must reference the same data set.

3. specify the key that you want to erase with the KEY= option and the ERASE= option in the INFILE statement. The ERASE= option specifies a numeric SAS variable that tells the SAS System whether a record is to be erased.

In most cases, when a record is retrieved for update, no user, including you, can access that particular record or any other records that are in the same control interval. Use the UPDATE= option to avoid user lockout when only a few of the records that are retrieved need to be updated (see “Using the UPDATE= Option” on page 31 for more information). For example, the program below reads records sequentially from the data set without retrieving them for update until the condition specified in the IF clause is met. When the IF condition is true (SASKEY = 547392749), the UPDATE= variable is set to 1, and the record is retrieved again with update access.

```sas
data keys;
    /* Use the SASKEY variable to select keys of records */
    /* to process. */
    infile myksds vsam keypos=kpvar keylen=klvar;
    retain kpvar klvar;
    input @kpvar saskey $varying200.klvar;
run;

    /* Update records in a KSDS */
    data updtksds;
        set keys;
        updtevar=0;
        infile myksds vsam key=saskey update=updtevar;
        input;
        if (saskey eq '547392749') then do;
            updtevar=1;
            input;

            /* Assign a value to _INFILE_, which contains the */
            /* update data. */
            file myksds vsam;
            put @1 _infile_ @10 'Flintstone Fred ';
        end;
run;
```
Refer to Table 6.1 on page 56 for a list of the options that you can use to erase records. The following list explains which values you can set for the ERASE= option as well as what the values specify:

- When you set the ERASE= variable to a value of 1 before a PUT statement for the data set executes, the record is erased. Notice that the record is not updated with the data in the PUT statement; it is erased instead. However, for a KSDS, you must copy the key of the record to the PUT buffer by issuing an _INFILE_ argument in the PUT statement to identify the record.

  After a record is erased, the ERASE= variable is automatically reset to 0. Therefore, you must set it to 1 again to erase another record. This prevents you from inadvertently deleting a series of records.

- When you set the ERASE= variable to a value of 0 before a PUT statement for the data set executes, the record is updated with the data that is specified instead of being erased. This is the default action taken if you do not use the ERASE= option.

In the following example, the variable SASKEY in the SAS data set KEYS contains the keys of the records that you want to erase. Notice that the PUT statement erases the record rather than updating it because the ERASE= variable, ERASEVAR, is set to a value of 1.

```sas
data eleven;
  set keys;
  erasevar=1;
  infile myksds vsam key=saskey erase=erasevar;
  file myksds vsam;
  input;
  if (saskey eq '547392749') then do;
    put _infile_;
  end;
run;
```

### Combined Operations on a KSDS

You may want to perform more than one operation on a KSDS in one DATA step. For example, perhaps you want to read some records, update other records, and add new records in one DATA step. Regardless of the number of operations, you need only one pair of INFILE and FILE statements for the entire DATA step. Specify the VSAM option in both the INFILE and the FILE statements. Specify any other options that you might need to process the KSDS in its INFILE statement.

The following sections describe how the SAS System determines whether you want to add new records or update existing KSDS records.

### Adding Records without Reading

When you do not execute an INPUT statement before the PUT statement (because you are adding records without reading from the KSDS), the SAS System assumes that the data in the PUT statement is to be added as a new record. You must specify a new primary key in the PUT statement data.

If a record with this key already exists, VSAM refuses to replace it and returns a logical error with a feedback code of 8. To replace the existing record with the new data, set the KEY= variable to match the PUT statement key data, read the record with an INPUT statement, and re-execute the PUT statement. Remember that VSAM does not allow you to change the primary key field.
Key Testing with FEEDBACK= and the PUT Statement

You can use the FEEDBACK= option to test whether or not a record with a particular key exists. Then you can either update or add a record based on the value of the FEEDBACK= variable. The FEEDBACK= option specifies a SAS variable that is set to the VSAM logical error code when a logical error occurs. (See Chapter 9, “Error-Handling Techniques and Error Messages,” on page 93 for more information.)

The general key-testing technique using the FEEDBACK= option and the data in the PUT statement is as follows:

- When the FEEDBACK= variable is 0 after the PUT statement executes, the data in the PUT statement has been added as a new record.
- When the FEEDBACK= variable is 8 after the PUT statement executes, a record with that key already exists. Therefore, the data in the PUT buffer is not added as a new record because VSAM does not allow duplicate primary keys.

To replace the existing record, reset the FEEDBACK= and _ERROR_ variables to 0, set the KEY= variable to match the PUT statement key data, issue an INPUT statement, and re-execute the PUT statement.
For example,

```sas
data twelve;
  length keyvar $9. ;
  infile myksds vsam feedback=fdbk   key=keyvar keypos=poskey;
  file myksds vsam;

  /* Assign a value to the KEYVAR variable, */
  /* which contains the record’s key. */
  keyvar='964514789';
  lastname='Flintstone ';
  frstname='Fred ';
  address='1234 Quarry Rd';
  city='Boulder ';
  state='CO';
  zip='12345 ';
  balance='00999';
  gpa='1.33';
  class='SE';
  hrs='13';
  finaid='Y';

  /* Try to write as a new record (that is, without reading). */
  put @poskey keyvar $9.
      @10 lastname $10.
      @20 frstname $10.
      @30 address $15.
      @55 city $15.
      @70 state $2.
      @72 zip $5.
      @77 balance $5.
      @82 gpa $4.
      @86 class $2.
      @88 hrs $2.

  /* If the record already exists, reset FDBK and _ERROR_ */
  /* to 0, read in the record, write the record’s key, and */
  /* update the record with new data. */
  if fdbk=8 then do;
    fdbk =0;
    _error_  = 0;
    input;
    put  @ poskey keyvar $9.
         @10 lastname $10.
         @20 frstname $10.
         @30 address $15.
         @55 city $15.
         @70 state $2.
         @72 zip $5.
         @77 balance $5.
         @82 gpa $4.
         @86 class $2.
         @88 hrs $2.
```

Adding Records after Reading

When you are reading from the KSDS before you write, the SAS System assumes that the data that is in the PUT buffer is to modify the record that you have just read. This is true unless you have changed the primary key with PUT @ statements after an INPUT statement and before the final PUT statement executes.

When you have changed the primary key after an INPUT statement and before the PUT statement for the data set executes, the data in the PUT buffer is added as a new record as long as the key field does not duplicate the key of an existing record. A VSAM logical error occurs if the key duplicates the key of an existing record.

Key Testing with FEEDBACK=, KEY=, and the INPUT Statement

You can use the FEEDBACK= option to test whether or not a record with a particular key exists. You then can either update or add a record based on the value of the FEEDBACK= variable.

As mentioned earlier, the FEEDBACK= option specifies a SAS variable that is set to the VSAM logical error code when a logical error occurs. (See Chapter 9, “Error-Handling Techniques and Error Messages,” on page 93 for more information.)

The general key-testing technique using the FEEDBACK= and KEY= options and an INPUT statement is as follows:

- When the FEEDBACK= variable is 0 after the INPUT statement executes, a record with a key that matches the value of the KEY= variable has been found and is read into the input buffer.

- When the FEEDBACK= variable is 16 after the INPUT statement executes, a record with a key that matches the value of the KEY= variable does not exist.

To add the data as a new KSDS record, reset the FEEDBACK= and _ERROR_ variables to 0 and issue a PUT statement with the value of the KEY= variable in the key field location.

For example,

data thirteen;
  length keyvar $9.;
  infile myksds vsam feedback=fdbk key=keyvar keypos=poskey;

  /* Assign a value to the KEYVAR variable, */
  /* which contains the record’s key */
  keyvar='984312769';
  lastname='Rubble';
  frstname='Barney';
  address='1234 Gravel Rd';
  city='Boulder';
  state='CO';
  zip='12345';
  balance='00001';
  gpa='0.33';
  class='SE';
hrs='13';
finaid='Y';

input;

/* If there is no record with this key, reset the FDBK and */
/* _ERROR_ variables to 0, and write a message on the SAS */
/* print file that a new record has been added with this key. */

if fdbk=16 then do;
  fdbk =0;
  _error_ = 0;
  file print;
  put 'New record added. Key is ' keyvar;
end;

/* Write the record to the data set: we are updating if there */
/* is a record with this key and adding a new record if */
/* there is not. */

file myksds vsam;
put @poskey keyvar $9.
  @10 lastname $10.
  @20 firstname $10.
  @30 address $15.
  @55 city $15.
  @70 state $2.
  @72 zip $5.
  @77 balance $5.
  @82 gpa $4.
  @86 class $2.
  @88 hrs $2.
  @90 finaid $1.;

stop;
run;

Comparing Key-testing Techniques

Notice the differences between the two key-testing techniques outlined in this section:

- The first technique is based on key data in the PUT statement and automatically adds the information as a new record if the key does not already exist. Be aware that you may create a record that you do not want.
- The second technique is based on an INPUT statement and the KEY= option. This method is safer because you must deliberately issue a PUT statement with the key field data in order to add a new record.
Introduction

Chapter 1 introduced three types of data set organization in VSAM:

- RRDS
- ESDS
- KSDS.

This chapter deals extensively with processing an RRDS in SAS jobs. Recall from Chapter 1 that an RRDS is a string of fixed-length slots, each identified by a relative record number. Each slot either contains a record or is empty. Records are stored and retrieved by the relative record number of the slot. An RRDS is appropriate for many applications that use fixed-length records or when the record number has a contextual meaning that can be used as a key.

This chapter explains how you can read, add, update, and erase RRDS records in SAS programs. The options that are associated with each of these operations are described in this chapter. In many cases, the option's meaning depends on how it is used within a SAS program. (Loading RRDS records is discussed separately in Chapter 4, “Defining and Loading the Sample VSAM Data Sets,” on page 41.)

This chapter presents two helpful tables: Table 7.1 on page 74 lists the options that are significant for processing RRDS records, and Table 7.2 on page 77 summarizes the access types for RRDS operations. In addition, there are examples of reading, adding, updating, and performing combined operations on an RRDS data set. The examples are
based on the STUDENT data set that is described in Appendix 2, “Sample STUDENT Data Set,” on page 107. You can run the examples by using the sample programs that are provided in the online help system and by following the steps to define and load an RRDS that are described in Chapter 4, “Defining and Loading the Sample VSAM Data Sets,” on page 41.

For information on using an ESDS in SAS jobs, refer to Chapter 5, “Processing an ESDS in a SAS Job,” on page 47. For information on using a KSDS in SAS jobs, refer to Chapter 6, “Processing a KSDS in a SAS Job,” on page 55.

Special SAS Options Used with an RRDS

The special SAS options fall into functional categories. The following table lists the special SAS options and categories that are used for processing an RRDS data set. Informational and record retrieval options are specified in the INFILE statement. Record storage options are specified in the FILE statement.

Table 7.1  SAS Options for an RRDS

<table>
<thead>
<tr>
<th>Functional Category</th>
<th>Option</th>
<th>Informational</th>
<th>Record Retrieval</th>
<th>Record Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BACKWARD</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BUFND=</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BUFNI=</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONTROLINTERVAL</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ERASE=</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>ERRORABEND</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FEEDBACK=</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PASSWD=</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RRN=</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>RC4STOP</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RECORDS=</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RESET</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>SEQUENTIAL</td>
<td></td>
<td>X*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SKIP</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UPDATE=</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VSAM</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

* Meaningful only if you also have a PUT statement.

Reading Records from an RRDS

You can read RRDS records with sequential access, direct access, and with a combination of both sequential and direct access (see Table 7.2 on page 77). The type of
RRDS read operation is specified with the appropriate options in the SAS INFILE statement. You must specify either the VSAMREAD or the VSAMUPDATE global SAS system option in order to read VSAM data sets.

**Reading an RRDS with Sequential Access**

With sequential access, the RRDS records are read in relative record order; that is, they are read from the first record to the last. This is the default.

For example,

```sas
data one;
  infile myrrds vsam;
  input;
  ...more SAS statements...
```

If the BACKWARD option is specified, the data set is read backward, starting with the last record and ending with the first.

**Reading an RRDS with Direct Access**

An RRDS is read directly using keyed direct access where the relative record number (RRN) is treated as a key. For this type of access to be meaningful, you must know the RRNs of the records that you want to read. The RRN might be known if it has some relationship to the record contents or if you have obtained it, for example, from the _RRN_ automatic variable in a previous SAS DATA step.

To read an RRDS with keyed direct access, use the RRN= option in the INFILE statement. The RRN= option defines a SAS variable whose value you set to the RRN of the record that you want the SAS System to read. The variable is created if it does not exist and is not added to the output data set.

For example, in the following program the RRDS data set is read sequentially, and the relative record numbers are obtained from the automatic variable _RRN_ and stored in the SAS data set RRNS. DATA TWO uses the RRNS SAS data set to process the RRDS by relative record number.

```
data rrns;
  infile myrrds vsam ;
  input;
  rrnvar=_rrn_ ;
  keep rrnvar;
run;

data two;
  set rrns;
  infile myrrds vsam rrn=rrnvar;
  input;
  ...more SAS statements...
```

**Reading an RRDS with Skip Sequential Access**

With skip sequential access, the initial record of a series is located with keyed direct access. Once the first record is obtained, subsequent records are retrieved sequentially.
Skip sequential processing improves performance because sequential retrieval requires less overhead and is faster than direct retrieval. Skip sequential access is also useful when you know the RRN of the first record that you want but do not know (or do not want to specify) the RRN of subsequent records.

Use the SKIP option in the INFILE statement to specify skip sequential processing. Retrieve the first record directly by specifying the RRN of the record that you want with the RRN= option in the INFILE statement. With the SKIP option, leaving the value specified by the RRN= variable unchanged turns off direct access and indicates that subsequent records are to be retrieved with sequential access. The relative record number of each record that is retrieved is returned in the _RRN_ automatic variable. The relative record numbers might not be consecutive because some of the slots may be empty.

When you process skip sequentially, you must specify an explicit means of stopping the DATA step. In the example below, end-of-file sets the feedback code to 4, and the IF FDBK=4 clause stops the DATA step. Note that the SKIP option retrieves only the one record with an RRN that matches the value of the RRN= variable value. You must supply statements to read additional records.

The following example processes an RRDS skip sequentially. For meaningful output, this example assumes that the data set was sorted by class before it was loaded. The program reads in the RRNUMS data set, reads all the records in the PROCESS data set that have those RRNs, and then writes them to a SAS print file. Note that the SKIP option retrieves only the records that are identified by the RRNs. You must supply statements to read additional records. In the following example, the program sequentially reads other records in the same class:

```sas
data rruns;
  input idnum class $;
cards;
  0001 FR
  0013 JU
  0025 SO
;
run;

data process;
  set rruns;
  file print;
  if _n_=1 then do;
    put 'The RRDS records selected skip sequentially are: ';
    put;
  end;
  /* Get the first record wanted with direct access. */
  infile myrrds vsam rrn=idnum skip feedback=fdbk;
    input @;
  /* Stop if the FEEDBACK= variable indicates end-of-file */
  /* or if the RRN slot is empty or invalid. */
  if fdbk=4 | fdbk=16 | fdbk=192 then do;
    _error_=0;
    if fdbk=4 then stop;
  else do;
    put 'RRN slot is empty, or invalid RRN. The feedback ';
    'code is ' fdbk ' and RRN is ' idnum;
    fdbk =0;
```
/* Read next records sequentially while the class matches. */

/* Write the records to the SAS print file. */

input @86 classnow $ 86-87;
do while (classnow=class);
   put _infile_;

   /* Stop if the FEEDBACK= variable indicates end-of-file */
   /* or if the RRN slot is empty or invalid. */
   if fdbk=4 | fdbk=16 | fdbk=192 then do;
      _error_=0;
      if fdbk=4 then stop;
      else do;
         put 'RRN slot is empty, or invalid RRN. The feedback ' code is ' fdbk;
         fdbk=0;
         return;
      end;
   end;
   input @86 classnow $ 86-87;
end;
run;

Table 7.2 Access Types for RRDS Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Read (INFILE/INPUT Statements)</th>
<th>Write (FILE/PUT Statements)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Sequential</td>
<td>Does not apply</td>
</tr>
<tr>
<td></td>
<td>Direct with RRN= option</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Skip sequential with SKIP and RRN= options</td>
<td></td>
</tr>
<tr>
<td>Add*</td>
<td>Direct with RRN= option</td>
<td>Must be direct: use the RRN= option</td>
</tr>
<tr>
<td></td>
<td>Sequential with SEQUENTIAL and RRN= options</td>
<td></td>
</tr>
<tr>
<td>Update</td>
<td>Sequential</td>
<td>Direct: the record read is the record updated</td>
</tr>
<tr>
<td></td>
<td>Direct with RRN= option</td>
<td></td>
</tr>
<tr>
<td>Erase</td>
<td>Sequential</td>
<td>Direct: the record read is the record erased</td>
</tr>
<tr>
<td></td>
<td>Direct with RRN= option</td>
<td></td>
</tr>
<tr>
<td>Load</td>
<td>Does not apply</td>
<td>Sequential: in relative record order</td>
</tr>
<tr>
<td></td>
<td>Direct with RRN= option</td>
<td>Direct with RRN= option</td>
</tr>
</tbody>
</table>

* The INPUT statement is not required.
Adding Records to an RRDS

To add records to an RRDS:
1. specify the SAS system option VSAMUPDATE.
2. include both an INFILE and a FILE statement for the data set. Specify the VSAM option in both the INFILE and the FILE statements. Specify all other options in the INFILE statement, which must precede the FILE statement.
3. use the PUT statement to write the record.

For a list of options that you can use when adding records, see Table 7.2 on page 77. When you add records to an existing RRDS, you do not have to include an INPUT statement with the INFILE statement. An INPUT statement is unnecessary because you do not have to read a record in order to add a new record.

Access Type When Adding Records

In an RRDS, records are added with direct access (see Table 7.2 on page 77). You must supply the relative record number by using the RRN= variable in the INFILE statement. The slot that is specified by the RRN= variable must be vacant in order to add a new record to that location in the data set.

Adding Records While Reading

If you are adding new records while reading the RRDS sequentially, use the SEQUENTIAL option in the INFILE statement. The SEQUENTIAL option specifies sequential retrieval that is combined with direct record storage. Use the SEQUENTIAL option only when you add new RRDS records while sequentially reading existing records. Use both the SEQUENTIAL and the RRN= options in the INFILE statement to indicate that the RRN= variable specifies the record that is to be added rather than the record that is to be read. For example,

```sas
data addrns;
    /* Create a new RRDS record and assign it to the */
    /* NEWREC variable. */
    length newrec $90.;
    id='984312769';
    lastname='Rubble ';
    firstname='Barney ';
    address='1234 Gravel Rd ';
    city='Boulder ';
    state='CO';
    zip='12345';
    balance='00001';
    gpa='0.33';
    class='SE';
    hrs='13';
    finaid='Y';
    newrec = id||lastname||firstname||address||city||state||
       zip||balance||gpa||class||hrs||finaid;
    /* Assign the RRN of the new record (NEWREC) to */
    /* the ADDRRN variable. */
```
Processing an RRDS in a SAS Job

Adding Records While Reading

```sas
addrrn=31;
run;

data four;
  set addrrns;
  n = 0;

  /* Read the RRDS records with sequential access. */
  infile myrrds vsam rrn=addrrn sequential;
  input;
  n = n+1;

  /* Add the new RRDS record with direct access: RRN=ADDRRN option */
  /* applies to writing when the SEQUENTIAL option is used. The */
  /* NEWREC variable contains the complete new record. */
  if (n=1) then do;
    file myrrds vsam;
    put newrec;
  end;
run;

In the example, a record for a new student, BARNEY RUBBLE, is defined and added
to data set MYRRDS. The data set is read sequentially, but the record is added with
direct access to the record slot that is identified by the ADDRRN variable.

If you are adding new records while reading the RRDS directly, use the RRN= option
to specify both the that record you want to read and the slot number where you want to
add a record. First, set the RRN= variable to the record that you want to read. Read
the record, but before the PUT statement executes, reset the RRN= variable to the slot
number where you want to insert the new record. Change the value of the RRN= variable after you retrieve the record with an INPUT statement but before you write with a PUT statement.

The following example uses direct access to read and add new records. As in the
previous example, the NEWREC variable in the SAS data set ADDRRNS contains the
complete new record, and the ADDRRN variable contains the RRN where the record is
to be added.

```sas
    data addrrns;

    /* Create a new RRDS record and assign it to the */
    /* NEWREC variable. */
    length newrec $90.;
    id='995613769';
    lastname='Rubble ';
    firstname='Bettie ';
    address='1234 Gravel Rd ';
    city='Boulder ';
    state='CO';
    zip='12345';
    balance='00001';
    gpa='2.22';
    class='SE';
    hrs='13';
    finaid='Y';
    newrec = id||lastname||firstname||address||city||state||
Updating Records in an RRDS

To update records in an RRDS:

1. specify the SAS system option VSAMUPDATE.
2. include both an INFILE and a FILE statement for the data set. Specify the VSAM option in both the INFILE and the FILE statements. Specify all other necessary options in the INFILE statement, which must precede the FILE statement.
3. use an INPUT statement to read the record that is being modified. You must first retrieve the record sequentially or by direct access using the RRN= option before you can update the data set.
4. use the PUT statement to write the complete record.

For a list of the options that you can use when you update records, refer to Table 7.1 on page 74.

When you update a record in an RRDS, input access for reading can be either sequential or direct (see Table 7.2 on page 77). Access is sequential unless the RRN= direct access option is specified in the INFILE statement. Sequential access in an RRDS means in relative record order. You can use a combination of direct and sequential access to the input data set if you specify the SKIP option in the INFILE
statement. See the earlier section, “Reading an RRDS with Skip Sequential Access” on page 75, for more information.

When you update an RRDS record, you must include a PUT statement to write the complete record. There are two common ways of writing the record with the PUT statement:

- Build the complete record by specifying all fields with the PUT statement. This method may be best when many of the fields need to be updated.
- Overlay certain fields in a copy (_INFILE_) of the existing record. This method is best when relatively few fields need to be updated.

The latter method is the easier for most applications. For example, the following statement copies the last record that is read into the PUT buffer and overlays the information starting in columns 10 and 30 with the values in NEWDATA1 and NEWDATA2:

```plaintext
put @ 1 _infile_
  @ 10 newdata1
  @ 30 newdata2;
```

When a record is retrieved for update, no user, including you, can access that particular record or any other records in the same control interval. Use the UPDATE= option to avoid user lockout when only a few of the records that are retrieved need to be updated. See “Using the UPDATE= Option” on page 31 for more information. In the following example, the RRDS records are read sequentially without being retrieved for update until the IF clause condition is met. When the IF condition is true (in this case, IDNUM=15), the UPDATE= variable is set to 1, and the record is retrieved again with update access.

```plaintext
data rrrnumbrs;
  /* Use the IDNUM variable to select the RRNs of */
  /* records to process. */
  infile myrrds vsam;
  input;
  idnum = _rrn_;
run;

data six;
set rrrnumbrs;
updtevar =0;
infile myrrds vsam rrr=idnum update=updtevar;
input;
if (idnum=15) then do;
  updtevar=1;
  input;
  /* Create the NEWDATA variable which contains */
  /* the update data. */
  newdata=36;
file myrrds vsam;
  put @ 1 _infile_ @88 newdata;
end;
run;
```
Erasing Records from an RRDS

To erase a record from an RRDS:

1. specify the SAS system option VSAMUPDATE.
2. use an INFILE statement and an INPUT statement to read the record and a FILE statement and a PUT statement to erase the record. The INFILE statement and FILE statement must have the same fileref; they must reference the same data set.
3. specify the record that you want to erase with the RRN= option and the ERASE= option in the INFILE statement. The ERASE= option specifies a numeric SAS variable that tells the SAS System whether a record is to be erased.

Refer to Table 7.1 on page 74 for a list of the options that you can use when you erase records. The following list explains which values you can set for the ERASE= option as well as what the values specify:

- When you set the ERASE= variable to a value of 1 before a PUT statement for the data set that executes, the record is erased. Notice that the record is not updated with the data in the PUT statement; it is erased instead. However, for an RRDS, you must still copy the relative record number of the record to the PUT buffer by issuing an _INFILE_ argument in the PUT statement in order to identify the record.

  After a record is erased, the ERASE= variable is automatically reset to 0. Therefore, you must set it to 1 again in order to erase another record. This prevents the inadvertent deletion of a series of records.

- When you set the ERASE= variable to a value of 0 before a PUT for the data set executes, the record is updated with the data that is specified instead of being erased. This is the default action taken if the ERASE= option is not used.

In the following example, the variable RRNVAR in the SAS data set ERASEREC contains the RRNs of the records that you want to erase. Notice that the PUT statement erases the record rather than updating it because the ERASE= variable, ERASEVAR, is set to a value of 1.

```sas
data seven;
  set rrnumbrs;
  erasevar=1;
  infile myrrds vsam rrn=rrnvar erase=erasevar;
  file myrrds vsam;
  input;
  put _infile_;
run;
```

Combined Operations on an RRDS

You might want to perform more than one operation on an RRDS in one DATA step. For example, perhaps you want to read some records, update other records, and add new records in one DATA step. Regardless of the operations, you need only one pair of INFILE and FILE statements for the entire DATA step. Specify the VSAM option in both the INFILE and the FILE statements. Specify any other options that you need to process that RRDS in its INFILE statement.

The following sections describe how the SAS System determines whether you want to add new or update existing RRDS records.
Adding Records without Reading

When you do not execute an INPUT statement before the PUT statement (because you are adding records without reading from the RRDS), the SAS System assumes that the data in the PUT statement is to be added as a new record, provided that you have specified an empty relative record slot with the RRN= option. The slot that is specified by the RRN= variable must be vacant in order to add a new record in that location of the data set.

If the slot already contains a record, VSAM refuses to replace it and returns a logical error with a feedback code of 8. The FEEDBACK= option can be used to determine whether a particular slot is empty.

Slot Testing with FEEDBACK=, RRN=, and the PUT Statement

You can use the FEEDBACK= option to test whether the relative record slot that is specified by RRN= is empty. You can then either update or add a record based on the value of the FEEDBACK= variable. The FEEDBACK= option specifies a SAS variable that is set to the VSAM logical error code when a logical error occurs. (See Chapter 9, “Error-Handling Techniques and Error Messages,” on page 93 for more information.)

The general slot-testing technique using the FEEDBACK= and RRN= options and the data in the PUT statement is outlined as follows:

- When the FEEDBACK= variable is 0 after the PUT statement executes, the slot is empty and the data in the PUT buffer has been added as a new record.
- When the FEEDBACK= variable is 8 after the PUT statement executes, the slot is not empty.

To update the existing record, reset the FEEDBACK= and _ERROR_ variables to 0, read the record with an INPUT statement, and re-execute the PUT statement.

If you want to add a new record rather than replace the one in the slot, change the RRN= variable to another slot number and re-execute the PUT statement.

For example,

data rrdstest;
   length lastname $10 frstname $10 class $2;
   input id lastname frstname class;
   datalines;
   15 FLINTSTONE FRED SE
   30 RUBBLE BARNEY SO
   31 FLINTSTONE WILMA SE
   32 RUBBLE BETTIE SO
 ;

data eight;
   set rrdstest;
   infile myrrds vsam feedback=fdbk rrn=id;
   file myrrds vsam;

   /* Assume this is a new record and write it without reading. */
   put @10 lastname $10.
   @20 frstname $10.
   @86 class $2.;

   /* If the FEEDBACK= variable indicates that the record */
Adding Records after Reading

When you read from the RRDS before you write, the SAS System assumes that the data in the PUT statement modifies the record that you have just read unless you change the RRN= variable value before the PUT statement executes.

When you have changed the RRN= variable after an INPUT statement and before the PUT statement for the data set executes, the data in the PUT buffer is added as a new record (if the changed RRN= value specifies a vacant slot).

Slot Testing with FEEDBACK=, RRN=, and the INPUT Statement

You can use the FEEDBACK= option to test whether the relative record slot that is specified by RRN= is empty. You can then either update or add a record based on the value of the FEEDBACK= variable. As mentioned in the previous section, the FEEDBACK= option specifies a SAS variable that is set to the VSAM logical error code when a logical error occurs. (See Chapter 9, “Error-Handling Techniques and Error Messages,” on page 93 for more information.)

The general slot-testing technique using the FEEDBACK= and RRN= options and the INPUT statement is outlined as follows:

- When the FEEDBACK= variable is 0 after the INPUT statement executes, the record that is in the slot specified by RRN= has been read into the input buffer. Execute a PUT statement in order to update the record.
- When the FEEDBACK= variable is 16 after the INPUT statement executes, the slot that is specified by RRN= is empty. Reset the FEEDBACK= and _ERROR_ variables to 0, and execute a PUT statement in order to add the PUT buffer data as a new record in this slot.

For example,

```sas
data rrdsinfo;
  /* Select values for lastname, firstname, and class. */
  length lastname $10 firstname $10 class $2;
  input id lastname firstname class;
  datalines;
  15 FLINTSTONE FRED SE
  30 RUBBLE BARNEY SO
  31 FLINTSTONE WILMA SE
  32 RUBBLE BETTIE SO
;...
```

```sas
data nine;
  set rrdsinfo;
  infile myrrds vsam feedback=fdbk rrn=id;
  /* Read the relative record number to be updated. */
```
input;
file myrrds vsam;

/* If the FEEDBACK= variable indicates that the relative */
/* record slot number is empty, reset the FDBK and */
/* _ERROR_ variables to 0, and write a new record. */
if fdbk=16 then do;
  fdbk=0;
  _error_=0;
  put @10 lastname $10.
    @20 firstname $10.
    @86 class $2.;
end;
end;

/* If the FEEDBACK= variable indicates PUT for update */
/* without previous INPUT then reset the FDBK and */
/* _ERROR_ variables to 0, and write the new record. */
if fdbk=92 then do;
  fdbk=0;
  _ERROR_=0;
  put @10 lastname $char10.
    @20 firstname $char10.
    @86 class $char2.;
end;

/* If the record exists, update the class field. */
else do;
  put _infile_ @86 class;
end;
run;

Comparing Slot-testing Techniques

Notice the differences between the two slot-testing techniques outlined in this section:

- The first, based on data in the PUT statement and the RRN= option, automatically adds the information as a new record if the slot is empty. Be aware that you may create a record that you do not want.

- The second, based on an INPUT statement and the RRN= option, is a safer technique because you must deliberately issue a PUT statement to add a new record.
CHAPTER 8

Using Alternate Indexes for VSAM Data Sets

Introduction

This chapter explains how to create alternate indexes for VSAM data sets. An alternate index provides another way to access a VSAM data set, as described in “Keyed direct access with an alternate index” on page 12.

You can create an alternate index for an ESDS or a KSDS. An alternate index is itself a KSDS that references the base cluster.

The examples in this chapter illustrate how to create an alternate index for both a new data set and an existing data set. The alternate index enables you to perform direct keyed access of the STUDENT data set by state code. Refer to Appendix 2, “Sample STUDENT Data Set,” on page 107 for the data set that is used in the examples.

On both CMS and OS/390, you define and build alternate indexes by using IBM Access Method Services (AMS). Under CMS, you must additionally assign logical names for the VSAM catalog, data set, and alternate index before you execute the AMSERV job. This step is not necessary under OS/390.

Creating an Alternate Index for an ESDS

The examples in this section show how to create an alternate index for an ESDS under the CMS and OS/390 operating environments.

The following example shows how to create an alternate index for an ESDS under CMS. To create an alternate index under CMS, you must assign logical names for the VSAM catalog, data set, and alternate index before you execute the AMSERV job.

The following CMS EXEC procedure issues DOS DLBL statements in order to assign the logical names:

/* */
'DLBL IJSYSCT V DSN mycat (PERM'
'DLBL myesds V DSN dsname.esds.student ( VSAM'
'DLBL mypath V DSN dsname.esds.student ( VSAM'

The following IBM AMS AMSERV job defines the alternate index data set:
CREATE AIX (NAME(dsname.ESDS.MYINDEX) -
RELATE(dsname.ESDS.STUDENT) -
RECSZ(20 37) -
SHAREOPTIONS(2,3) -
VOLUMES(xxxx) -
REUSE -
NONUNIQUEKEY -
RECORDS(10 5) -
CISZ(2048) -
KEYS(2,69) -
CATALOG(mycat)
)

DELETE dsname.ESDS.STUDENT.PATH

DEFINE PATH (NAME(dsname.ESDS.STUDENT.PATH) -
PATHENTRY(dsname.ESDS.MYINDEX) -
CATALOG(mycat)
)

The first qualifier of the data set name, dsname, represents a value that the user supplies. The user also supplies the volume number (xxxx).

The following SAS DATA step reads the data from the ESDS in alternate index order:

data aixtest;
  infile mypath vsam;
  input id $9. lastname $10. frstname $10. address $25. city $15.
    state $2. zip $5. balance $5. gpa $4. class $2. hrs $2.
    finaid $1.;
run;

The following example shows the IBM AMS IDCAMS job that is needed to create an alternate index for an ESDS under OS/390. On OS390, an IDCAMS job can be submitted via JCL or in a SAS macro variable. The following is an example of TSO DEFINE commands in a macro variable. The example, “Creating an Alternate Index for an Existing KSDS” on page 89, shows how to submit an IDCAMS job from JCL.

An alternate index cannot be created if the ESDS is created with the REUSE statement.

/* Remove alternate index if it exists. */
x "delete (’dsname.esds.myindex’) purge alternateindex";

/* Define an alternate index in the cluster ‘dsname.esds’. */
/* Use the two-letter state code as the alternate index key. */
%let def = %str(define aix %name(’dsname.esds.myindex’) );

/* Relate the index to the cluster entry STUDENT. */
%let def=%def %str(relate(’dsname.esds.student’ ));

/* Specify the record size. */
%let def=%def %str(recsz(19 19 ) );
%let def=%def %str(shareoptions(2,3 ) );
Using Alternate Indexes for VSAM Data Sets

Creating an Alternate Index for an Existing KSDS

The following example creates an alternate index over an existing KSDS by using IDCAMS via JCL. If the data set already has an alternate index that is defined, it is erased and then redefined. After the alternate index is built, the SAS System is invoked to read the data set via the alternate index and to write the records to the SAS print file.

//DALTINDEX JOB accounting information
// *
// * Define an alternate key for an existing KSDS.
// *
Creating an Alternate Index for an Existing KSDS

Chapter 8

//STEP1 EXEC PGH=IDCAMS
//SYSPRINT DD SYSOUT=A

/*
/* If an alternate index already exists, delete it.
/* Then define the alternate index.
/*
//SYSIN DD *
DELETE (dsname.KSDS.STUDENT.ALTINDEX) PURGE ALTERNATEINDEX
IF LASTCC=8 THEN SET MAXCC=0

DEFINE ALTERNATEINDEX (name(dsname.KSDS.STUDENT.ALTINDEX) -
  KEYS(2 69) VOLUMES(xxxx) RECSZ(34 34) -
  RELATE(dsname.KSDS.STUDENT) UPGRADE -
  REUSE -
  NONUNIQUEKEY -
  CISZ(2048) -
  RECORDS(10 5))
IF MAXCC=0 THEN -
  DEFINE PATH (NAME(dsname.KSDS.STUDENT.PATH) -
    PATHENTRY(dsname.KSDS.STUDENT.ALTINDEX ))
IF MAXCC=0 THEN -
  BLDINDEX INDATASET(dsname.KSDS.STUDENT ) -
    OUTDATASET(dsname.KSDS.STUDENT.ALTINDEX )

/*
/*
/* Invoke the SAS System to read the data set via the alternate index
/* defined in STEP1.
/*
//STEP2 EXEC SAS,PARM=' VSAMREAD '
//SYSUDUMP DD SYSOUT=A
//PATH DD DISP=SHR,DSN=dsname.KSDS.STUDENT.PATH
//SYSIN DD *

/* Read the KDS via the alternate key. Write the records */
/* to the SAS print file, putting the observation number */
/* before each observation. */

data one;
  infile path;
  input;
  file print;
  put _n_ @5 _infile_;
/*
/*

To access the data set by the alternate index, you must have a DD statement that
references the data set name in the DEFINE PATH statement. Also note that the
STEP2 EXEC statement that invokes the SAS System specifies the SAS system option
VSAMREAD, which is needed only if your installation's default value for this option is
NOVSAMREAD.
Calculating Record Size

The AMS RECORDSIZE parameter requires the average and maximum record size, in bytes, of the alternate index record. When you calculate the maximum record size, if the alternate index record spans control intervals, the RECORDSIZE parameter can be larger than the CONTROLINTERVALLSIZE. Use the following formula to calculate the maximum record size of spanned records:

\[ \text{MAXLRECL} = \frac{\text{CI}}{\text{CA}} \times (\text{CISZ} - 10) \]

MAXLRECL is the maximum spanned record size. CI/CA represents the number of control intervals per control area. CA is the number of control areas. CISZ is the quantity control interval size.

Use the following formulas to determine the average size of the alternate index record when the alternate index supports

an ESDS:

\[ \text{RECSZ} = 5 + \text{AIXKL} + (n \times 4) \]

a KSDS:

\[ \text{RECSZ} = 5 + \text{AIXKL} + (n \times \text{BCKL}) \]

where

- AIXKL is the alternate-key length (see the KEYS parameter)
- BCKL is the base cluster’s prime-key length (you can issue the AMS LISTCAT command to determine the base cluster’s prime-key length)
- \( n=1 \) when the UNIQUEKEY parameter is specified (in which case, RECSZ is also the maximum record size)
- \( n \) is equal to the number of base cluster records that have the same alternate index key when NONUNIQUEKEY is specified.

In the preceding examples in this chapter, the average record size for alternate key STATE was calculated as follows:

- \( 5 + 2 + (3 \times 4) = 19 \) for the ESDS
- \( 5 + 2 + (3 \times 9) = 34 \) for the KSDS

Specifying the same value for average and maximum identifies the records as fixed length. Refer to the appropriate IBM documentation for more information on calculating record size.
Introduction

This chapter introduces you to the types of errors that can occur when you use the SAS System to process VSAM data sets. The types of errors are physical and logical. The chapter begins by explaining the differences between the two types of errors and then it discusses error detection and error-handling techniques, the FEEDBACK= variable and other error detection variables, and VSAM feedback codes. Table 9.2 on page 99 shows how COBOL status key values correspond to VSAM feedback codes.

Physical Errors

A physical error (also known as an I/O error) occurs when VSAM is unable to access a data set or record because of a hardware error. A hardware error is usually (but not always) caused by a problem with the disk on which the VSAM data set resides. Physical errors are very rare.

When a physical error occurs while processing VSAM data sets, the SAS System:

1. prints a set of appropriate messages in the SAS log.
2. sets the _ERROR_ automatic variable to 1.
3. fills the logical record buffer with blanks. If a physical error occurs while reading, the INPUT buffer is filled with blanks. If it occurs while writing, the PUT buffer is filled with blanks.
4. sets the _IORC_ automatic variable to 12, which is the VSAM return code for physical errors.
5. sets the _FDBK_ automatic variable to the VSAM feedback code for the physical error.
6. continues with the DATA step.
Logical Errors

Logically errors result from mistakes in program logic. There are two types of logical errors that can occur when you process VSAM data sets in a SAS program:

- SAS logical errors, which the SAS System detects before invoking VSAM
- VSAM logical errors, which VSAM detects while it attempts to process a request from your SAS program.

### SAS Logical Errors

The SAS VSAM interface looks for logical errors before it invokes VSAM. When the SAS System detects an error condition, it is a SAS logical error. When the SAS System cannot pass a request on to VSAM because of an error in your program, the DATA step terminates, and an error message that describes the error is printed on the SAS log.

The following conditions are examples of SAS logical errors:

- you attempt to update a VSAM data set without using the same fileref for both the INFILE and FILE statements.
- you try to erase an ESDS record by specifying the ERASE= variable on an INFILE statement that references an ESDS.
- you try to retrieve a spanned KSDS record by RBA.

By default, the standard INFILE statement options MISSOVER and STOPOVER are in effect for VSAM data sets, and they relate to SAS logical error conditions.

- The MISSOVER option assigns missing values to all variables in the INPUT statement that do not have values in the INPUT buffer.
  Note that the MISSOVER option allows processing to continue instead of terminating the DATA step.
- The STOPOVER option is the default when data overflows the current record when writing to a VSAM data set. The STOPOVER option causes
  - partially built records to be written to the data set
  - the DATA step to terminate immediately with an error message.

For more information on these options, refer to Chapter 3, “Processing VSAM Data Sets in SAS Programs,” on page 25.

### VSAM Logical Errors

Errors in your program logic that VSAM detects are called VSAM logical errors. Some common VSAM logical errors include

- trying to read a record that does not exist
- trying to update a record without reading it first
- trying to create a new record that violates VSAM restrictions.

When a VSAM logical error is encountered, the following automatic SAS variables are set:

1. The _IORC_ variable is set to a value of 8. (_IORC_ contains the value of the VSAM input/output return code.)
2. _FDBK_ is set to the VSAM feedback code. Some of the _FDBK_ values depend on the type of operating system and the VSAM release in use at your installation. However, the most common values are the same for all operating systems. For
Err or-Handling Techniques and Error Messages △ Using the FEEDBACK= Option 95

more information about the VSAM logical error and feedback codes, refer to your IBM documentation.

VSAM sets the I/O return code and the feedback code and returns their values to the SAS VSAM interface. The interface makes these values available to your SAS program in the automatic SAS variables _IORC_ and _FDBK_.

Error-Handling Techniques

How FEEDBACK= Differs from _IORC_ and _FDBK_

The FEEDBACK= option specifies a SAS variable that is set to the VSAM feedback code. The variable is set only when VSAM encounters a logical error. That is, the variable’s value is 0 until a logical error occurs. The nonzero value indicates what kind of logical error was detected. “Some Common Causes of Logical Errors” on page 97 describes the feedback codes that are most likely to be returned in the FEEDBACK= variable.

Note that both the _FDBK_ and the FEEDBACK= variables are set to the VSAM feedback code when a logical error occurs. The distinction between the two values is that only by specifying the FEEDBACK= variable (and resetting it) can you continue to process and detect later errors that may occur. The ability to reset the FEEDBACK= variable after taking appropriate action to handle the error is very significant and is discussed below. For this reason, it is strongly recommended that you use the FEEDBACK= option for all VSAM data sets in which logical errors may occur.

Other distinctions are that _FDBK_ is also set

- when VSAM detects a physical error
- even when VSAM sets a zero return code in certain situations.

You get a nonzero _FDBK_ with a zero _IORC_, for example, when you try to create a duplicate key in an alternate index.

The FEEDBACK= variable has a nonzero value only when a logical error occurs.

Using the FEEDBACK= Option

As mentioned in the previous section, the FEEDBACK= option in the INFILE statement specifies a SAS variable that is set to the VSAM feedback code when VSAM detects a logical error. You can determine what caused the error by inspecting the FEEDBACK= variable value. You can then design program logic that takes appropriate action depending on the value of the FEEDBACK= variable. You must reset the values of both the FEEDBACK= variable and the _ERROR_ variable to 0 in order to continue processing.

Resetting the variable to 0 allows you to continue processing in a meaningful way. That is, you can continue both to read and write records and detect other errors in the DATA step. If you do not reset the FEEDBACK= and _ERROR_ variables before the next INPUT or PUT statement, the SAS System assumes that your program cannot handle the error condition, and it executes the following:

1 prints a message that includes information about the data set and the VSAM logical error code on the SAS log.

2 terminates the DATA step.
The DATA step also terminates when the FEEDBACK= option is not specified, and a logical error occurs while it attempts to write with a PUT statement.

You must use the FEEDBACK= option to use the key-testing techniques for a KSDS (described in Chapter 6, “Processing a KSDS in a SAS Job,” on page 55) and the slot-testing techniques for an RRDS (described in Chapter 7, “Processing an RRDS in a SAS Job,” on page 73).

VSAM cannot return data to the input buffer when there is a logical or physical I/O error. Subsequent INPUT statements cannot read from an empty INPUT buffer, which leaves variables without values. To avoid this situation, test the values of _IORC_ and the FEEDBACK= variable by using a trailing @ with the INPUT statement that initiates the VSAM read request. For example,

```
infile indata vsam feedback=NOERROR;
input @; /* Read: look at values of FEEDBACK= variable */
   /* and _IORC_. If OK, finish reading values */
   /* into variables and write them to the SAS */
   /* print file. */
if _IORC_ = 0 and NOERROR=0 then do;
   input var1 $ var2 var3 $;
   file print;
   put var1 var2 var3;
end; /* If _IORC_ and NOERROR=0 */
else if _IORC_ = 12 then do;
   /* Physical error has occurred. */
   /* INPUT buffer is empty: nothing to read. */
   _ERROR_ = 0; /* Reset the _ERROR_ variable. */
   file log; /* Write message on the SAS log. */
   put 'Physical error has occurred for observation ' _N_ '. ' /*
      'I/O return code is ' _IORC_ '.';
   input; /* Ignore blank buffer: release trailing @. */
   return;
end; /* Else: _IORC_=12 */
else if NOERROR ^= 0 then do;
   /* Logical error has occurred. */
   /* INPUT buffer is empty: nothing to read. */
   _ERROR_ = 0;
   file log; /* Write message on the SAS log. */
   put 'Logical error has occurred for observation ' _N_ '. '
      'Feedback code is ' noerror '.';
   NOERROR=0; /* Reset FEEDBACK= variable back to 0. */
   input; /* Ignore blank buffer: release trailing @ */
   _ERROR_ = 0; /* Above INPUT stmt. sets both the _ERROR_ */
   NOERROR=0; /* and the FEEDBACK= variables. Both need */
   /* to be reset to 0 again. */
   return;
end; /* Else: NOERROR ^= 0 */
...more SAS statements...
```

Using the INPUT @ statement gives you the opportunity to examine the FEEDBACK= variable for a nonzero value, which indicates that a logical error has occurred. If both the _IORC_ and the FEEDBACK= variables are zero, continue with the INPUT statement to read data into variables.

Notice that the _ERROR_ and the FEEDBACK= variable, NOERROR, need to be reset to 0 twice when set to a nonzero value by an INPUT statement with a trailing @.
They need to be reset to 0 the first time in order to continue processing. The processing continues by releasing the held record from the input buffer with an INPUT statement without a trailing @. This sets the _ERROR_ and FEEDBACK= variables to nonzero values again; therefore, they need to be reset to 0 a second time in order to proceed.

You may want to print error messages warning that either a physical error was encountered (if _IORC_ is 12) or a logical error was encountered (if the FEEDBACK= variable is not 0). You may also design logic to handle specific, anticipated FEEDBACK= variable values.

### Some Common Causes of Logical Errors

Below is a list of the VSAM feedback codes that are most likely to be returned in the FEEDBACK= variable. The error condition that is associated with each feedback code is briefly described. The codes in this list represent decimal values, and they are the same for all operating systems.

IBM manuals describe many VSAM feedback codes that are not returned to your SAS program. This is because the SAS VSAM interface looks for many error conditions before it passes requests to VSAM. A VSAM feedback code cannot be returned when the SAS System detects an error before it invokes VSAM. Instead, the SAS System prints a message that describes the error on the SAS log and stops the DATA step.

You get VSAM logical errors and, therefore, VSAM feedback codes if

- the SAS System cannot detect the error in advance (for example, user lockout)
- the SAS System does not know what action to take (for example, record not found).

Check the return codes as previously outlined and design your programs to take appropriate action for the various error conditions.

<table>
<thead>
<tr>
<th>Feedback Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>An end of data set was encountered (during sequential or skip sequential retrieval), or the search argument is greater than the high key of the data set.</td>
</tr>
<tr>
<td>8</td>
<td>You attempted to store a duplicate alternate key for an alternate index with the unique key option, or you attempted to store a record with a duplicate primary key. (For an ESDS accessed through an alternate index or a KSDS.)</td>
</tr>
<tr>
<td>12</td>
<td>Records were not in key sequence when they are required to be. You are probably trying to load the file out of key order. VSAM requires a KSDS to be loaded in key order. (For a KSDS.)</td>
</tr>
</tbody>
</table>
| 16            | Record not found. This means that you attempted one of two things:
- You tried to retrieve a record with a key that does not exist in the file. (For a KSDS.)
- You tried to retrieve a record with a relative record number that corresponds to an empty slot. (For an RRDS.) Also see feedback code 192. |
Feedback Code | Error Description
--- | ---
20 | User lockout occurred because someone else is concurrently accessing the file and has exclusive use of the control interval that you need. This feedback code is also returned if you read a record and then try to add a new record to the same control interval. (You can avoid this situation by specifying an UPDATE=0 before you read the record.)
32 | You have requested a record by RBA, and there is no record with the address given by the RBA= variable. (For a KSDS or an ESDS.)
36 | Key ranges were specified for the data set when it was defined, and the record you want to add has a key that is not within one of those key ranges. (For a KSDS.)
72 | You attempted to access only the data portion of the VSAM cluster.
88 | A request was issued for which VSAM was not properly positioned. This error code is almost always the result of lost positioning that is due to a previous logical error.
96 | You attempted to change either the primary key or the key of reference while updating a record. This error occurs only if you access a KSDS through an alternate index and attempt to change the primary key while updating a record. (For a KSDS.)
If you change the primary key while using it to access a KSDS, or if you change the key of reference while accessing the data set through an alternate index, the SAS System assumes that you intend for the record to be a new record (if the new key is not a duplicate).
108 | You tried to write a record that is too small to contain the full key. (For a KSDS.) Your SAS program probably has
- not copied the input record to the PUT buffer with a PUT _INFILE_: @ statement
- not built the key in the PUT buffer when creating a new record
- not built the key in the correct position in the record.
A good way to ensure that the key is in the correct position is to use the variable specified by the KEYPOS= option.
192 | You have specified an invalid relative record number with the RRN= variable. An invalid RRN is one that does not represent a slot within the file. If you specify the RRN of an existing but empty slot, the feedback code is 16 instead of 192. (For an RRDS.)

COBOL Status Key Values and VSAM Feedback Codes

Table 9.2 on page 99 contains the COBOL status key values that correspond to the VSAM feedback codes that were previously discussed. Before you invoke VSAM, the SAS System traps many error conditions that set status key values in COBOL programs. Therefore, there are many COBOL status key values that contain VSAM feedback codes that have no counterpart in SAS programs.
Table 9.2  COBOL Status Key Values and VSAM Feedback Codes

<table>
<thead>
<tr>
<th>VSAM Feedback Code</th>
<th>COBOL Status Key Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>22</td>
</tr>
<tr>
<td>12</td>
<td>21</td>
</tr>
<tr>
<td>16</td>
<td>23</td>
</tr>
<tr>
<td>20</td>
<td>93</td>
</tr>
<tr>
<td>32</td>
<td>90</td>
</tr>
<tr>
<td>36</td>
<td>92</td>
</tr>
<tr>
<td>88</td>
<td>21</td>
</tr>
<tr>
<td>96</td>
<td>94</td>
</tr>
<tr>
<td>108</td>
<td>92</td>
</tr>
<tr>
<td>192</td>
<td>23</td>
</tr>
</tbody>
</table>

For more information about COBOL status key values and their corresponding VSAM feedback codes, consult the appropriate IBM documentation.
Appendices

Appendix 1. System Dependencies 103

Appendix 2. Sample STUDENT Data Set 107
Introduction

The following sections describe the system dependencies for CMS when you process VSAM data sets. There are no special system dependencies for OS/390.

CMS System Dependencies

You can use the SAS System under CMS in order to read VSAM data sets that are created by the CMS and OS/390 operating systems. To process VSAM data sets with the SAS System under CMS, you need to have the following capabilities:

- Your installation must have the VSE VSAM interface for CMS.
- You must be able to access the disk volume containing the VSAM data set that you want to read as well as the disk volume that contains the VSAM catalog. These are not necessarily on different volumes.
- The VSAM data set cannot be cataloged in an ICF format catalog.
- Your installation must support shared disks (shared DASD). You cannot write to a VSAM data set on a disk that belongs to another operating system.

Your VSAM administrator can tell you if the VSE VSAM interface and shared DASD is supported, whether you can access the volumes that you need, and what kind of catalog is used. Consult the appropriate IBM documentation for additional information on using VSAM under CMS.

If the requirements outlined above are met, you can process VSAM data sets by following these steps. You must take these steps before the SAS System is invoked:

1. If the VSAM data set is on a volume owned by OS/390, issue the LINK and ACCESS commands to access the disk volume that contains the VSAM catalog and the VSAM data set that you want to process. You should use the following syntax for these commands:

   LINK userid virtual-address1 virtual-address2 RR
ACCESS virtual-address2 filemode

If the catalog and data set are on different disks, LINK and ACCESS commands are required for each disk.

2 Issue the following CMS commands:

SET DOS ON (VSAM)
ASSGN SYSCAT filemode
ASSGN SYSnnn filemode
DLBL IJSYSCT filemode DSN catalog-dataset (PERM SYSCAT)
DLBL fileref filemode DSN VSAM file-dataset (PERM SYSnnn)
SET DOS OFF

The first SET command places the virtual machine in the CMS/DOS environment. This must be done before the other CMS/DOS commands execute. The ASSGN SYSCAT and DLBL IJSYSCT statements define the VSAM catalog. The ASSGN and DLBL statements for SYSnnn define the VSAM dataset, where nnn is a number between 001 and 254. The second SET command returns the virtual machine to the regular CMS environment. You cannot continue with your SAS job with the DOS environment active. Consult the appropriate IBM documentation for descriptions of these CMS DOS commands.

Once you have entered the appropriate control language, invoke the SAS System and use the INFILE, FILE, INPUT, and PUT statements as you would to process any external data set. The VSAM type option should be specified in the INFILE and FILE statements. You should use the following syntax for these statements:

INFILE fileref VSAM;

FILE fileref VSAM;

For the SAS fileref, use the fileref assigned in the DLBL statement. The SAS System issues FILEDEFs for the VSAM file (fileref) and VSAM catalog (IJSYSCT) automatically, based on the information in the DLBL statements. If a FILEDEF with the same fileref is already in effect, it is replaced. The FILEDEFs are automatically cleared when the SAS session terminates.

Note: If you are using a version of the SAS System that is installed in segments (the SSEG option is in effect), be sure that the VSAM VSE interface segment and the SAS segments do not overlap. If they do overlap and have access to a version of the SAS System that is not in segments, specify the NOSSE option in the SAS invocation command or in the SAS configuration files. The SAS System abends if you attempt to use SAS segments that overlap with the VSE VSAM interface segment.

---

Reading a VSAM Data Set under CMS

This example shows the control language that is necessary for reading an OS/390 VSAM catalog and data set that are stored on the same disk. Only one LINK command and one ACCESS command are necessary.

LINK OSLINK 280 220 RR
ACCESS 220 D
SET DOS ON (VSAM)
ASSGN SYSCAT D
ASSGN SYS001 D
DLBL IJSYSCT D DSN VSA DATA CATALOG (PERM SYSCAT
DLBL MYVSAM D DSN DATA TST VSAMAB (PERM SYS001
SET DOS OFF
...invoke SAS and execute SAS programs...

Accessing a Multivolume Data Set under CMS

This example shows the control language that is necessary for a SAS program to access a multivolume VSAM data set. There are three LINK, ACCESS, and ASSGN commands because there are three disk volumes involved—one volume for the VSAM catalog and two volumes for the VSAM data set. However, only two DLBL commands are needed because only two data sets are used—the VSAM catalog data set and the VSAM data set (even though the VSAM data set spans two volumes). The MULT option informs CMS that the VSAM data set is multivolume, and the system prompts for the filemode and logical unit of the second (third, fourth, . . .) volume.

<table>
<thead>
<tr>
<th>Command</th>
<th>Parameters</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK</td>
<td>DOS 284</td>
<td>RR 284 RR</td>
</tr>
<tr>
<td>ACCESS</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>LINK</td>
<td>DOS 285</td>
<td>RR 285 RR</td>
</tr>
<tr>
<td>ACCESS</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>LINK</td>
<td>DOSBTO 347</td>
<td>RR 347 RR</td>
</tr>
<tr>
<td>ACCESS</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>SET</td>
<td>DOS ON</td>
<td>(VSAM)</td>
</tr>
<tr>
<td>ASSGN</td>
<td>SYSCAT E</td>
<td></td>
</tr>
<tr>
<td>ASSGN</td>
<td>SAS009 F</td>
<td></td>
</tr>
<tr>
<td>ASSGN</td>
<td>SYS019 G</td>
<td></td>
</tr>
<tr>
<td>DLBL</td>
<td>IJSYSCT E</td>
<td>DSN DOS VSAM WHOLE CATALOG (PERM SYSCAT</td>
</tr>
<tr>
<td>DLBL</td>
<td>MULTIV F</td>
<td>DSN DOS VSAM MULTIVOL (MULT SYS009 VSAM</td>
</tr>
<tr>
<td>SET</td>
<td>DOS OFF</td>
<td></td>
</tr>
</tbody>
</table>

CMS Restrictions on Write Access

In order to update a VSAM data set using the SAS System under CMS, you must have exclusive write access to the minidisk or minidisks that contain the data set and catalogs; otherwise, the integrity of the data is jeopardized. If you intend to update a VSAM data set under CMS, make sure that you do not link to any of the VSAM minidisks in MW (multiple write) mode.

Under CMS SAS, you can access both CMS and VSE VSAM data sets and catalogs. Usually, you cannot update a VSE VSAM data set from CMS. This is because the VSE operating system usually maintains write access to VSE VSAM data sets, which prevents you from obtaining the exclusive write access that is required to update. CMS VSAM data sets are generally the only VSAM data sets to which a CMS user can obtain exclusive write access.
The examples in this book use the fictional STUDENT data set that is shown next. If you want to run the examples, sample programs are provided that define and load an ESDS, KSDS, and RRDS with the student data on OS/390. You can then enter or copy the code for the other examples from the manual. The steps for defining and loading a VSAM data set are described in Chapter 4. The sample programs are available in the online help system and on the installation media; see your SAS Support Consultant for details.

**Output A2.1 STUDENT Data Set Used in ESDS, KSDS, and RRDS Examples**

<table>
<thead>
<tr>
<th>OBS ID</th>
<th>LASTNAME</th>
<th>FRSTNAME</th>
<th>ADDRESS</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Martin</td>
<td>Duanne</td>
<td>392 Hazelwood Dr.</td>
<td>New Hartford</td>
</tr>
<tr>
<td>3</td>
<td>Smith</td>
<td>Jerry</td>
<td>111 Lincoln Ave.</td>
<td>Boston</td>
</tr>
<tr>
<td>4</td>
<td>Allen</td>
<td>Nancy</td>
<td>423 Lakefront Dr.</td>
<td>Deerborne</td>
</tr>
<tr>
<td>5</td>
<td>Teague</td>
<td>Denise</td>
<td>556 Cherokee Rd.</td>
<td>Oklahoma City</td>
</tr>
<tr>
<td>6</td>
<td>Jones</td>
<td>Antony</td>
<td>110 Aberdeen Rd.</td>
<td>Albany</td>
</tr>
<tr>
<td>7</td>
<td>Friedman</td>
<td>Oscar</td>
<td>2845 Ocean Drive</td>
<td>Tampa</td>
</tr>
<tr>
<td>8</td>
<td>Cox</td>
<td>June</td>
<td>Rt. 2 Box 784</td>
<td>Cheyenne</td>
</tr>
<tr>
<td>9</td>
<td>Hawthorne</td>
<td>Jean</td>
<td>688 Ridge Rock Way</td>
<td>Bountiful</td>
</tr>
<tr>
<td>10</td>
<td>Doe</td>
<td>John</td>
<td>384 Main Street</td>
<td>Walla Walla</td>
</tr>
<tr>
<td>11</td>
<td>Taylor</td>
<td>Anne</td>
<td>47 Lawrence Circle</td>
<td>Lawrence</td>
</tr>
<tr>
<td>12</td>
<td>Hall</td>
<td>Brad</td>
<td>56 Starr Ave.</td>
<td>Phoenix</td>
</tr>
<tr>
<td>13</td>
<td>Starnes</td>
<td>Randy</td>
<td>1450 Rock Quarry Road</td>
<td>Lexington</td>
</tr>
<tr>
<td>14</td>
<td>Mitchell</td>
<td>Barbara</td>
<td>923 Kemper Court</td>
<td>Spartanburg</td>
</tr>
<tr>
<td>15</td>
<td>Ward</td>
<td>Keith</td>
<td>Box 2330 Hwy 90</td>
<td>Kalamazoo</td>
</tr>
<tr>
<td>16</td>
<td>Henson</td>
<td>Edward</td>
<td>783 12th Ave. Circle</td>
<td>Knoxville</td>
</tr>
<tr>
<td>17</td>
<td>Pierce</td>
<td>Timothy</td>
<td>1233 Hamilton Drive</td>
<td>Dallas</td>
</tr>
<tr>
<td>18</td>
<td>Thomas</td>
<td>Matthew</td>
<td>Rt. 4 Box 634</td>
<td>Reading</td>
</tr>
<tr>
<td>19</td>
<td>Thomas</td>
<td>Wanda</td>
<td>21 Martian Way</td>
<td>Jupiter</td>
</tr>
<tr>
<td>20</td>
<td>Miller</td>
<td>Frank</td>
<td>570 8th Avenue</td>
<td>New York</td>
</tr>
<tr>
<td>21</td>
<td>Jones</td>
<td>Tanya</td>
<td>289 Jones Street</td>
<td>San Diego</td>
</tr>
<tr>
<td>22</td>
<td>Bradley</td>
<td>Steve</td>
<td>PO Box 282</td>
<td>Albuquerque</td>
</tr>
<tr>
<td>23</td>
<td>Olsen</td>
<td>Wayne</td>
<td>PO Box 2580</td>
<td>Chicago</td>
</tr>
<tr>
<td>24</td>
<td>Allen</td>
<td>Beth</td>
<td>2834 Harcourt St.</td>
<td>Seattle</td>
</tr>
<tr>
<td>25</td>
<td>Miller</td>
<td>Carol</td>
<td>Rt. 3 Box 245</td>
<td>Jackson</td>
</tr>
<tr>
<td>26</td>
<td>Thomas</td>
<td>Henry</td>
<td>397 Pennsylvania Ave.</td>
<td>Washington</td>
</tr>
<tr>
<td>27</td>
<td>Allen</td>
<td>Harold</td>
<td>56 48th Street</td>
<td>Denver</td>
</tr>
<tr>
<td>28</td>
<td>Quimbley</td>
<td>Fred</td>
<td>934 Oak Street</td>
<td>Richmond</td>
</tr>
<tr>
<td>29</td>
<td>Hart</td>
<td>Jim</td>
<td>489 Hartford Drive</td>
<td>Miami</td>
</tr>
<tr>
<td>30</td>
<td>Hill</td>
<td>Thomas</td>
<td>2458 Johnson Parkway</td>
<td>Santa Monica</td>
</tr>
<tr>
<td>OBS</td>
<td>STATE</td>
<td>ZIP</td>
<td>BALANCE</td>
<td>GPA</td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
<td>-------</td>
<td>---------</td>
<td>-----</td>
</tr>
<tr>
<td>1</td>
<td>AR</td>
<td>83992</td>
<td>00050</td>
<td>2.13</td>
</tr>
<tr>
<td>2</td>
<td>CN</td>
<td>00103</td>
<td>00000</td>
<td>0.75</td>
</tr>
<tr>
<td>3</td>
<td>MA</td>
<td>00376</td>
<td>00025</td>
<td>4.00</td>
</tr>
<tr>
<td>4</td>
<td>MI</td>
<td>50471</td>
<td>00025</td>
<td>2.45</td>
</tr>
<tr>
<td>5</td>
<td>OK</td>
<td>53062</td>
<td>00000</td>
<td>2.63</td>
</tr>
<tr>
<td>6</td>
<td>NY</td>
<td>10025</td>
<td>00000</td>
<td>2.35</td>
</tr>
<tr>
<td>7</td>
<td>FL</td>
<td>34988</td>
<td>00035</td>
<td>1.22</td>
</tr>
<tr>
<td>8</td>
<td>NY</td>
<td>59334</td>
<td>00100</td>
<td>2.33</td>
</tr>
<tr>
<td>9</td>
<td>UT</td>
<td>79483</td>
<td>00050</td>
<td>2.75</td>
</tr>
<tr>
<td>10</td>
<td>WA</td>
<td>90126</td>
<td>00025</td>
<td>1.55</td>
</tr>
<tr>
<td>11</td>
<td>KS</td>
<td>69037</td>
<td>00000</td>
<td>3.22</td>
</tr>
<tr>
<td>12</td>
<td>AZ</td>
<td>82453</td>
<td>00000</td>
<td>3.46</td>
</tr>
<tr>
<td>13</td>
<td>KY</td>
<td>48223</td>
<td>00000</td>
<td>2.89</td>
</tr>
<tr>
<td>14</td>
<td>SC</td>
<td>29027</td>
<td>00050</td>
<td>2.98</td>
</tr>
<tr>
<td>15</td>
<td>MI</td>
<td>47893</td>
<td>00000</td>
<td>2.48</td>
</tr>
<tr>
<td>16</td>
<td>TN</td>
<td>37294</td>
<td>00150</td>
<td>2.93</td>
</tr>
<tr>
<td>17</td>
<td>TX</td>
<td>55934</td>
<td>00000</td>
<td>0.43</td>
</tr>
<tr>
<td>18</td>
<td>PA</td>
<td>16382</td>
<td>00025</td>
<td>3.33</td>
</tr>
<tr>
<td>19</td>
<td>FL</td>
<td>34892</td>
<td>00000</td>
<td>1.83</td>
</tr>
<tr>
<td>20</td>
<td>NY</td>
<td>10003</td>
<td>00100</td>
<td>1.95</td>
</tr>
<tr>
<td>21</td>
<td>CA</td>
<td>97274</td>
<td>00000</td>
<td>4.00</td>
</tr>
<tr>
<td>22</td>
<td>NM</td>
<td>73485</td>
<td>00020</td>
<td>3.80</td>
</tr>
<tr>
<td>23</td>
<td>IL</td>
<td>56038</td>
<td>00000</td>
<td>3.85</td>
</tr>
<tr>
<td>24</td>
<td>WA</td>
<td>91673</td>
<td>00000</td>
<td>2.37</td>
</tr>
<tr>
<td>25</td>
<td>MS</td>
<td>42865</td>
<td>00025</td>
<td>4.00</td>
</tr>
<tr>
<td>26</td>
<td>DC</td>
<td>19534</td>
<td>00000</td>
<td>2.50</td>
</tr>
<tr>
<td>27</td>
<td>CO</td>
<td>60049</td>
<td>00050</td>
<td>1.83</td>
</tr>
<tr>
<td>28</td>
<td>VA</td>
<td>20784</td>
<td>00070</td>
<td>3.90</td>
</tr>
<tr>
<td>29</td>
<td>FL</td>
<td>33134</td>
<td>00000</td>
<td>3.55</td>
</tr>
<tr>
<td>30</td>
<td>CA</td>
<td>99999</td>
<td>00000</td>
<td>4.00</td>
</tr>
</tbody>
</table>


Glossary

Access Method Services (AMS)

an IBM utility program that defines VSAM files, allocates space for them, converts ISAM files to VSAM files, modifies file attributes in the catalog, facilitates file portability between operating systems, creates backup copies of files and indexes, helps make inaccessible files accessible, and lists records and catalog entries.

addressed direct access

each record is stored and retrieved directly by its address relative to the beginning of the file (relative byte address), which is independent of the record’s location relative to data that is previously accessed. Addressed direct access can be used to access ESDS and KSDS records.

AIX

an abbreviation for alternate index.

alternate index

an index that is related to a given base cluster and is organized by an alternate key (a key other than the prime key of the associated base cluster data records). Its function is to provide an alternate method for locating records. An alternate index can be built over an ESDS or a KSDS. Use AMS (below) to build an alternate index.

AMS

see Access Method Services (AMS).

base cluster

the data component of an ESDS or the data and prime index components of a KSDS.

cluster

a named structure consisting of a group of related components. In a KSDS, a cluster consists of a data component and an index component. In an ESDS and an RRDS, the cluster consists of a data component only.

control area

a group of control intervals that compose the unit that VSAM preformats as records are added to the data set.

control interval

a contiguous area of secondary (disk) storage that VSAM uses for storing records and the control information that describes them. It is the unit of information that VSAM transmits to and from direct access storage. Although the size of control intervals
varies from one file to another, the size within a file is fixed, either by VSAM or the user (within VSAM-imposed restrictions). VSAM chooses the size based on the DASD type, record size, and smallest amount of virtual storage space that user application programs make available for I/O buffers.

data set
in VSAM, the major unit of data storage and retrieval. It is a collection of data in one of several prescribed arrangements, and it is described by control information to which the system has access. Data set is synonymous with file.

DATA step view
a type of SAS data set that contains a stored DATA step program. Like other SAS data views, a DATA step view contains a definition of data that is stored elsewhere. This view’s input data can come from several sources, including raw data files, other SAS files and data views, and VSAM data sets. Because a DATA step view only reads (opens for input) other files, you cannot update this view’s underlying data. DATA step views can function only as input data sets.

direct access
each record is stored and retrieved directly according to its logical address (key or relative record number) or its address relative to the beginning of the file (relative byte address). Data storage and retrieval depend only on the location of the data and not on a reference to data previously accessed. Direct access is synonymous with random access.

ESDS
a VSAM file type whose record sequence is determined by the order in which the records are entered into the file, without respect to the record contents. ESDS stands for entry-sequenced data set.

file
see data set.

fileref
a name that you temporarily assign to an external data set. Once a fileref (file reference name) has been assigned, you can use the fileref in the SAS System as a shorthand reference to the data set. For VSAM processing, the fileref stores data set and processing attributes specific to the operating environment so that the fileref can be used in the place of the host-dependent external data set name in SAS statements and commands.

index
in VSAM, a data set that relates each record’s key to its relative location in the file. VSAM uses the index to sequence and locate the records of a KSDS.

key
in VSAM, one or more consecutive characters, located in the key field of each record, that are used to identify the record and establish its order with respect to the other records.

keyed direct access
in VSAM, a method of access in which records are retrieved and stored by specifying the record’s key for a KSDS or the relative record number (RRN) for an RRDS.

key field
a field that contains the record’s key, which is located in the same position in each record of a KSDS.

KSDS
a VSAM file type in which record order is determined by the EBCDIC collating sequence of the key field contents. KSDS stands for key-sequenced data set.
**logical record**
data that is requested of or given to the data management function (VSAM in this case) as a unit.

**path**
in VSAM, a data set name that is used to access a VSAM file through an alternate index.

**physical record**
a unit of information that is stored on secondary (disk) storage. A physical record may consist of all or part of a logical record, and it may contain multiple logical records. Its form depends on the characteristics of the file and the disk type.

**prime key**
the main key of a key-sequenced base cluster. It is the key by which the KSDS records are initially entered and ordered. Each KSDS record must have a unique prime key.

**RBA**
the displacement of a record or control interval from the beginning of the file. RBA stands for relative byte address.

**RRDS**
a VSAM file type whose records are loaded into fixed-length slots and referenced by the record numbers of the slots. RRDS stands for relative record data set.

**RRN**
in an RRDS, a number that identifies the slot, or data space, and the record contained therein. RRN stands for relative record number.

**sequential access**
in VSAM, the retrieval and storage of data in either entry sequence for an ESDS, key sequence for a KSDS, or relative record sequence for an RRDS.

**skip sequential access**
in VSAM, a two-step process that combines both direct and sequential access. The initial record is located by keyed direct access, and subsequent records are retrieved sequentially. Skip sequential access can be used with a KSDS, an RRDS, and an ESDS that is accessed through an alternate index.

**spanned record**
a logical record that is contained in more than one control interval.

**stored record**
a data record, together with its control information, that is stored in auxiliary storage.

**VSAM**
an abbreviation for Virtual Storage Access Method, which is a multifunction, all-purpose IBM data access method.

**VSAM catalog**
a KSDS with an index that contains extensive file and volume information that VSAM requires to locate files, allocate and deallocate storage space, verify the authorization of a program or operator to gain access to a file, and to accumulate file usage statistics.
## Index

**A**

access methods
- addressed direct mode 12
- alternate index direct 12
- keyed direct mode 12
- sequential 14

adding records to a KSDS
- after reading to an existing data set 70
- without reading 64

adding records to an ESDS
- after reading 54
- to an existing data set 51

adding records to an RRDS
- access type for 78
- after reading 84
- to an existing data set 78
- while reading 78
- without reading 83

adding records to VSAM data sets
- addressed direct mode 12
- AIX access method 12
- alternate index access method 12
- alternate indexes, creating ESDS 87
- KSDS 89
- alternate key index access method 12
- approximate key retrieval for KSDS 89
- KEYGE option 21
- automatic variables _FDBK_ 16
- _IORC_ 16
- _RBA_ 16
- _RRN_ 16

**B**

BACKWARD (BKWD) option, INFILE or FILE statement 19
- base clusters 13
- BLKSIZE= option, INFILE statement 16
- blocksize, specifying buffers
  - current input 17
  - data, number of 19

**C**

carriage-control characters
- clusters 8
- base clusters 13
- CMS system dependencies
  - accessing multivolume data sets 107
  - reading VSAM data sets 104
  - write access restrictions 105
- CNV option, INFILE or FILE statement 19
- COBOL status key values, and VSAM feedback codes
  - COLUMN= option, INFILE statement 16
  - control areas 8
  - control intervals 8
  - reading 20
- CONTROLINTERVAL (CTLINTV) option, INFILE or FILE statement 20

**D**

DA (Direct Access Organization) data sets 4
- data integrity, VSAM data sets 30
- date/time values 61
- DELIMITER option, INFILE statement 16
- delimiters, list input
  - reading as characters 16
- specifying 16
- Direct Access Organization (DA) data sets 4
- DLBL statements 26
- DSD option, INFILE statement 16

**E**

END= option, INFILE statement
- end-of-dataset condition 16
- end-of-file condition 16
- end-of-record overflow 13
- end-of-volume condition 17

EOF= option, INFILE statement 16
- EOV= option, INFILE statement 17
- ERASE= option, INFILE or FILE statement 31
- erasing records
  - ERASE= option 20
  - ESDS 31
  - KSDS 31, 66
  - RRDS 31, 82
  - VSAM data sets 20, 31
- error codes
  - saving 20
  - VSAM feedback codes 97
- error handling
  - COBOL status key values, and VSAM feedback codes 98
  - _FDBK_ variable 95
  - FEEDBACK= option 95
  - hardware errors 93
  - I/O errors 93
  - _IORC_ variable 95
  - logical errors, causes of 97
  - logical errors, SAS programs 94
  - logical errors, VSAM programs 94
  - physical errors 93
  - VSAM feedback codes, and COBOL status key values 98
- VSAM feedback codes, table of 97
- ESDS 57
- access types 48
- adding records after reading 54
- adding records to existing data set 51
- alternate indexes, creating 87
- clusters 8
- combining DATA step operations 53
- control intervals and areas 8
- defining under CMS 42
- direct access
  - direct access by RBA 49
  - erasing records 31
- keyed direct access by alternate keys 50
- keyed direct access, specifying 20
- loading a new data set 45
- loading in a SAS DATA step 46
- lockout, avoiding 52
- RBA, setting 21
- reading records from
  - SAS options for sequential access 49

**Index** 113
Index

updating records 52
EXPANDTABS option, INFILE statement 17
external data sets 26
FILE statement
erasing records 31
loading VSAM data sets 45
SAS options for VSAM 19
sharing buffers with INFILE statement
FILENAME= option, INFILE statement 17
FILENAME statement, associating filerefs with VSAM data sets 26
filerefs, for VSAM data sets 22
FILEVAR= option, INFILE statement 17
FLOWOVER option, INFILE statement 17
GENKEY option 20
GENKEY option, INFILE or FILE statement 20
reading a KSDS 59
reading by approximate key 59
reading by RBA 62
reading with alternate index 61
reading with direct access 57
reading with keyed direct access 57
reading with sequential access 62
SAS options for 56
access types 58
adding records after reading 70
adding records to existing data set 64
adding records without reading 67
alternate indexes, creating 89
direct access types 58
SECONDARY option, INFILE statement 23
SECONDARY= option, INFILE statement 23
sequential access 57
lockout, avoiding 22
ESDS 52
KSDS 66
RRDS 81
UPDATe=, SAS option 22
logical errors
causes of 97
SAS programs 94
VSAM programs 94
logical record length 18
look-ahead read 29
LRECL= option, INFILE statement 18
LS= option, INFILE statement 17
MEANS procedure reports, generating (example) 33
MISSOVER option, INFILE statement 18
multivolume data sets, CMS system dependencies 105
N
N= option, INFILE statement 18
O
OBS= option, INFILE statement 18
K
key length
KEYLEN= option 21
KSDS 60
KEY= option, INFILE or FILE statement 20
key testing
reading a KSDS 58
key position 21
INPUT statement
PUT statement
keyed direct access
specifying
with alternate index 12
keyed direct mode 12
KEYGE option, INFILE or FILE statement 59
KEYLEN= option, INFILE or FILE statement 59
KEYPOS= option, INFILE or FILE statement 60
keys, definition 5
KSDS 5
MISSOVER option, INFILE statement 18
MISSING values 18
packed decimal data 61
PRINT procedure listings, generating (example) 32
RBA, setting 21
reading by approximate key 59
reading by RBA 62
reading with alternate index 61
reading with direct access 57
reading with keyed direct access 57
reading with sequential access 62
SAS options for 56
skip sequential access 22
updating records 65
updating with a windowing program (example) 34
upgrading records 52
EXPANDTABS option, INFILE statement 17
external data sets 26
FILE statement
erasing records 31
loading VSAM data sets 45
SAS options for VSAM 19
sharing buffers with INFILE statement
FILENAME= option, INFILE statement 17
FILENAME statement, associating filerefs with VSAM data sets 26
filerefs, for VSAM data sets 22
FILEVAR= option, INFILE statement 17
FLOWOVER option, INFILE statement 17
GENKEY option 20
GENKEY option, INFILE or FILE statement 20
reading a KSDS 59
reading by approximate key 59
reading by RBA 62
reading with alternate index 61
reading with direct access 57
reading with keyed direct access 57
reading with sequential access 62
SAS options for 56
access types 58
adding records after reading 70
adding records to existing data set 64
adding records without reading 67
alternate indexes, creating 89
direct access types 58
SECONDARY option, INFILE statement 23
SECONDARY= option, INFILE statement 23
sequential access 57
lockout, avoiding 22
ESDS 52
KSDS 66
RRDS 81
UPDATe=, SAS option 22
logical errors
causes of 97
SAS programs 94
VSAM programs 94
logical record length 18
look-ahead read 29
LRECL= option, INFILE statement 18
LS= option, INFILE statement 17
MEANS procedure reports, generating (example) 33
MISSOVER option, INFILE statement 18
multivolume data sets, CMS system dependencies 105
N
N= option, INFILE statement 18
O
OBS= option, INFILE statement 18
options
FILE statement options 19
for ESDS 48
for KSDS 56
for loading VSAM data sets 45
for RRDS 74
functional categories 23
INFILE statement options 16
SAS system options 15
SAS system options, default values 15
SAS system options, required for reading
VSAM data sets 28
VSAM options 24

P
PAD option, INFILE statement 18
padding input records with blanks 18
Partitioned Organization (PO) data sets 4
PASSWD= option, INFILE or FILE statement 21
password protection, VSAM data sets 21
Physical Sequential (PS) 4
PO (Partitioned Organization) data sets 4
PRINT option, INFILE statement 22
PRINT procedure listings, generating (example) 32
PS (Physical Sequential) 4
PUT statement
key testing 68
slot testing 83
updating ESDS records 52

R
random access method 11
reading VSAM data sets 29
RBA 12
getting latest 18
reading a KSDS 62
reading an ESDS 49
setting for an RRDS 21
RBA= option, INFILE or FILE statement
adding records to an ESDS 54
reading an ESDS 49
_RBA_ variable 16
RC4STOP option, INFILE or FILE statement
reading a KSDS
by approximate key 59
by RBA 62
direct access 57
GENKEY option 59
KEY= option 58
keyed direct access 57
KEYGE option 59
KEYLEN= option 60
KEYPOS= option 60
sequential access 57
skip sequential access with alternate index 62
reading an ESDS
access types 48
by RBA 60
RBA= option 49
reading an RRDS
by RNN 75
direct access 75
sequential access 75
skip sequential access 75
reading control intervals 20
reading VSAM data sets backward 19
CMS system dependencies
control intervals 104
default access 28
direct access enabling 15
first record, specifying 17
last record, specifying 18
list input delimiters, reading as characters 20
SAS system options required sequential access 28
unbuffered 18
READPW= option, INFILE or FILE statement 22
RECFM= option, INFILE statement 18
record format 18
record length limit 17
record size, calculating 91
record structure and organization 8
RECORDS= option, INFILE or FILE statement 22
RECORDSIZE parameter 91
RECORGE= option, FILENAME statement
RESET option, INFILE or FILE statement
RRDS
adding records after reading 64
adding records to existing data set 64
adding records while reading 64
adding records without reading 63
combining DATA step operations 62
control intervals and areas defining under CMS 42
erasing records 31
loading a new data set 46
loading in a SAS DATA step 46
lockout, avoiding 61
options for 74
RBA, setting 21
reading with direct access 75
reading with sequential access 75
slot testing 83
updating records 80
RRN 12
getting the latest 16
reading RRDS 75
setting 22
UPDATE= option, INFILE or FILE statement
RRDS
adding records after reading 64
adding records to existing data set 64
adding records while reading 64
adding records without reading 63
combining DATA step operations 62
control intervals and areas defining under CMS 42
erasing records 31
loading a new data set 46
loading in a SAS DATA step 46
lockout, avoiding 61
options for 74
RBA, setting 21
reading with direct access 75
reading with sequential access 75
slot testing 83
updating records 80
RRN= option, INFILE or FILE statement
getting the latest 16
reading RRDS 75
setting 22
UPDATE= option, INFILE or FILE statement
VSAMUPDATE system option 15
with a windowing program (example) 34

V
Virtual Storage Access Method 1

S
SAS automatic variables
_FDBK_
_IOCRC_
_RBA_
_RRN_
SAS system options 15
default values 15
for reading VSAM data sets 28
SCANOVER option, INFILE statement 18
sequential access method 11
loading VSAM data sets 45
reading VSAM data sets 28
specifying 22
SEQUENTIAL option, INFILE or FILE statement 22
SHAREBUFFERS option, INFILE statement 18
sharing data sets during update 50
SKIP option, INFILE or FILE statement 22
skipping a sequential access method
reading VSAM data sets 29
specifying 22
slot testing 83
comparison of techniques 83
INPUT statement 84
PUT statement 83
spanned records 8
START= option, INFILE statement 18
STOPOVER option, INFILE statement 18
STUDENT data set 107

T
tab characters, expanding 17
time/date values 61
truncated input records 18
TRUNCOVER option, INFILE statement 15

U
UNBUFFERED option, INFILE statement 18
UPDATE= option, INFILE or FILE statement
updating VSAM data sets 31
updating VSAM data sets 30
data integrity 30
enabling 15
ESDS 52
KSDS 54
limitations on lockout, avoiding 22
losing updates 30
record specification for 22
record update status 31
RRDS 50
sharing data sets 30
UPDATE= option 22
VSAMUPDATE system option 15
with a windowing program (example) 34
VSAM catalogs, defining 41
VSAM data sets 5
adding records 20
approximate key retrieval 21
approximate key retrieval, specifying 59
buffers, sharing carriage-control characters closing 17
CMS system dependencies 107
column pointer location 16
combined DATA step operations comparison of 32
carriage-control characters 18
closing 17
current input buffer 17
data buffers, number of determining type of differences between end-of-dataset condition 16
end-of-file condition 16
end-of-volume condition 17
erasing records 21
error codes, saving error when opening 22
external data sets fileref, specifying 22
first record, specifying generic-key processing 22
IBM equivalent data set types 5
index buffers, number of input blocksize 16
key length 21
keyed direct access key position 21
line length 17
line location 18
lines, number of list input delimiters, reading as characters 18
loading, enabling 15
lockout, avoiding 22
logical record length 18
missing values name of current number of records, getting opening operations, and access types operations, types of padding with blanks password protection RBA, setting reading control intervals reading, backward reading, enabling reading, specifying first record reading, specifying last record reading, unbuffered record format record length limit referring to resetting to empty RRN, setting scanning to a specified character sequential access skip sequential access tab characters, expanding truncated records writing, specifying first column number
VSAM data sets, defining under CMS 42
ESDS 42
KSDS 42
RRDS 42
VSAM catalogs 41
VSAM data sets, defining under OS/390 43
VSAM data sets, loading existing data set in a SAS DATA step new data set options for access types
VSAM data sets, reading default access direct access look-ahead read required system options sequential access skip sequential access
VSAM data sets, updating data integrity enabling ESDS 52 KSDS 65 limitations on 30 lockout, avoiding losing updates 30 record update status RRS 80 sharing data sets specifying records for UPDATE= option 30 VSAMUPDATE system option 15 with a windowing program (example) 34
VSAM feedback codes 97 and COBOL status key values _FDBK_ variable 16 table of 97
VSAM I/O return code, variable for 16
VSAM option, INFILE or FILE statement 22
VSAM options 24 VSAM, definition 4 VSAMLOAD system option 15 VSAMREAD system option 15 VSAMUPDATE system option 15
W
WRITEPW= option, INFILE or FILE statement 23
writing VSAM data sets, specifying first column number 18
If you have comments or suggestions about *SAS® Guide to VSAM Processing, Version 8*, please send them to us on photocopy of this page or send us electronic mail.
For comments about this book, please return the photocopy to
SAS Institute
Publications Division
SAS Campus Drive
Cary, NC 27513
email: yourturn@sas.com
For suggestions about the software, please return the photocopy to
SAS Institute
Technical Support Division
SAS Campus Drive
Cary, NC 27513
email: suggest@sas.com
Visit our Web page located at www.sas.com/pubs

You will find product and service details, including

- sample chapters
- tables of contents
- author biographies
- book reviews

Learn about

- regional user-group conferences
- trade-show sites and dates
- authoring opportunities
- custom textbooks

Explore all the services that SAS Institute Publishing has to offer!

Your Listserv Subscription Automatically Brings the News to You
Do you want to be among the first to learn about the latest books and services available from SAS Institute Publishing? Subscribe to our listserv newdocnews-l and, once each month, you will automatically receive a description of the newest books and which environments or operating systems and SAS release(s) that each book addresses.

To subscribe,

1. Send an e-mail message to listserv@vm.sas.com.
2. Leave the “Subject” line blank.
3. Use the following text for your message:
   
   subscribe NEWDOCNEWS-L your-first-name your-last-name

   For example: subscribe NEWDOCNEWS-L John Doe
Create Customized Textbooks Quickly, Easily, and Affordably

SelecText® offers instructors at U.S. colleges and universities a way to create custom textbooks for courses that teach students how to use SAS software.

For more information, see our Web page at www.sas.com/selectext, or contact our SelecText coordinators by sending e-mail to selectext@sas.com.

You’re Invited to Publish with SAS Institute’s User Publishing Program

If you enjoy writing about SAS software and how to use it, the User Publishing Program at SAS Institute offers a variety of publishing options. We are actively recruiting authors to publish books, articles, and sample code. Do you find the idea of writing a book or an article by yourself a little intimidating? Consider writing with a co-author. Keep in mind that you will receive complete editorial and publishing support, access to our users, technical advice and assistance, and competitive royalties. Please contact us for an author packet. E-mail us at sasbbu@sas.com or call 919-677-8000, then press 1-6479. See the SAS Institute Publishing Web page at www.sas.com/pubs for complete information.

See Observations®, Our Online Technical Journal

Feature articles from Observations®: The Technical Journal for SAS® Software Users are now available online at www.sas.com/obs. Take a look at what your fellow SAS software users and SAS Institute experts have to tell you. You may decide that you, too, have information to share. If you are interested in writing for Observations, send e-mail to sasbbu@sas.com or call 919-677-8000, then press 1-6479.

Book Discount Offered at SAS Public Training Courses!

When you attend one of our SAS Public Training Courses at any of our regional Training Centers in the U.S., you will receive a 15% discount on book orders that you place during the course. Take advantage of this offer at the next course you attend!

SAS Institute
SAS Campus Drive
Cary, NC 27513-2414
Fax 919-677-4444

E-mail: sasbook@sas.com
Web page: www.sas.com/pubs
To order books, call Fulfillment Services at 800-727-3228*
For other SAS Institute business, call 919-677-8000*

* Note: Customers outside the U.S. should contact their local SAS office.