THE POWER TO KNOW.

SAS® 9.3 VSAM Processing for z/OS

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc 2011. *SAS® 9.3 VSAM Processing for z/OS*. Cary, NC: SAS Institute Inc.

SAS® 9.3 VSAM Processing for z/OS

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

1st electronic book, July 2011

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

Contents

What's New in SAS 9.3 VSAM Processing for z/OS

Overview

SAS 9.3 VSAM Processing for z/OS has enhancements for using record-level sharing (RLS).

New VSAM Options for the FILE and INFILE Statements

- RLS | NORLS option specifies record-level sharing (RLS) to open an RLS-eligible data set. For more information, see RLS | NORLS on page 25.
- RLSREAD option enables you to specify the level of read integrity required for an RLS-eligible data set (INFILE only). For more information, see RLSREAD on page 25.

Recommended Reading

- *SAS Companion for z/OS*

For a complete list of SAS publications, go to support.sas.com/bookstore. If you have questions about which titles you need, please contact a SAS Publishing Sales Representative:

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-3228
Fax: 1-919-677-8166
E-mail: sasbook@sas.com
Web address: support.sas.com/bookstore

Chapter 1

The Virtual Storage Access Method (VSAM)

What is Virtual Storage Access Method (VSAM)?

Introduction to VSAM

VSAM is an IBM data access method that enables you to *organize* and *access* records in a disk data set. VSAM is available under the z/OS operating environment. There are three types of data set organization:

- Entry-Sequenced Data Set (ESDS)
- Key-Sequenced Data Set (KSDS)
- Relative-Record Data Set (RRDS)

VSAM has three types of *access* to records in VSAM data sets:

- sequential
- direct
- skip sequential

In addition, VSAM provides the following access and retrieval options:

- 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)

SAS 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 4.1 on page 30 for a summary of the operations that SAS 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. For more information, see "Direct Access" on page 7 . (The terms *direct* and *random* are sometimes used interchangeably when referring to data organization, access methods, and storage devices. SAS documentation uses the term *direct*, but you might find that *random* is used in other literature.)

Sequential access means that you retrieve a series of records in sequence. Sequence has a different meaning for each of the three VSAM data set organizations. For more information, see "Sequential Access" on page 7.

Skip sequential access means that you use a combination of both direct and sequential access. For more information, see "Skip Sequential Access" on page 9.

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:

- Physical Sequential (PS)
- Partitioned Organization (PO)
- Indexed Sequential (IS)
- Direct Access (DA)
- Virtual Storage Access Method (VSAM)

VSAM data sets can be one of the following:

- Entry-Sequenced Data Set (ESDS)
- Key-Sequenced Data Set (KSDS)
- Relative-Record Data Set (RRDS)

In each data set type except VSAM, the records are organized in a unique way, depending on their purpose. Each type of data set organization has one or more special access methods. (For example, a data set that uses DA organization is characterized by a predictable relationship between the key of a record and the address of that record on a DASD device.) The programmer establishes this relationship and must supply most of the logic required to locate the individual records.

VSAM is a multifunction, all-purpose access method. VSAM is different from the other data set types because it provides a functional equivalent for most of the other data set organizations, as follows:

- 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 types of data set organizations that you access with VSAM differ from others for two reasons:

- They are device independent.
- They 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 for 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 suitable 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 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.

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.

The figure below, Figure 1.1 on page 4 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.

The following figure summarizes the differences between the three VSAM data set types.

Figure 1.1 *VSAM Data Set Organization: Data Components and Index Components*

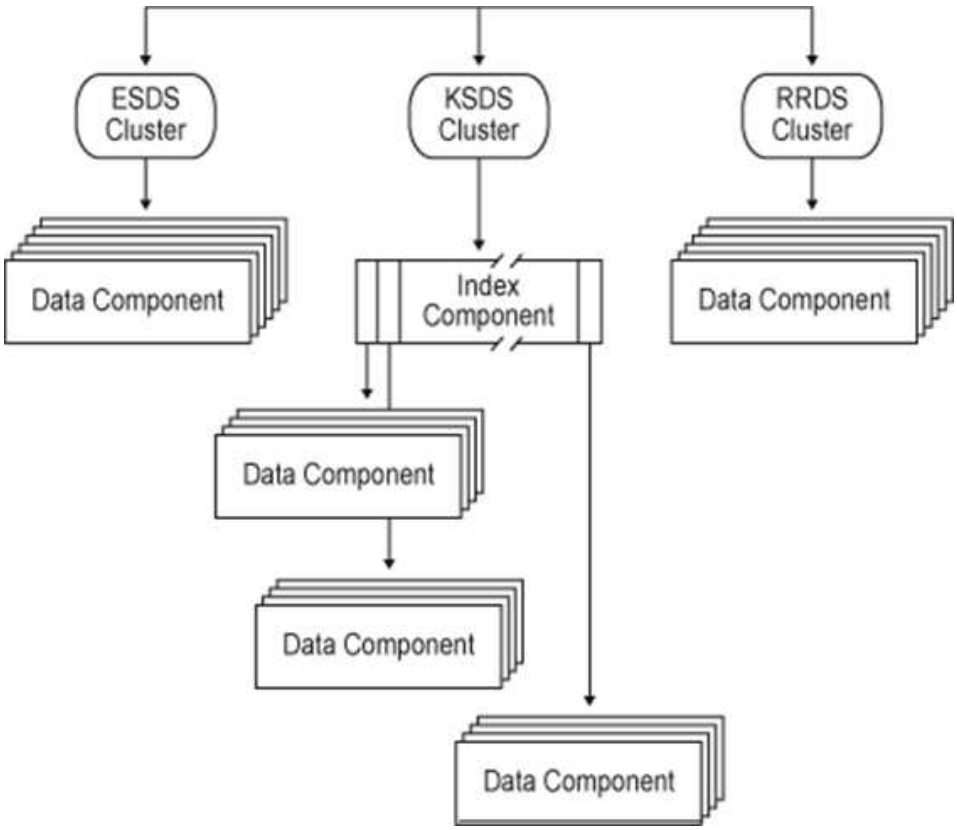


Table 1.1 *Comparison of VSAM Data Set Types*

	ESDS	KSDS	RRDS
What is the method for sequential access?	Entry order	Primary key order	RRN**
What is the method for direct access?	RBA**	Key RBA	RRN
What are the types of record format?	Fixed Variable Spanned	Fixed Variable Spanned	Fixed
Is record length changeable?	No	Yes	No

	ESDS	KSDS	RRDS
Where are new records added?	End of file	Anywhere	RRN slot (if empty)
Is embedded free space defined?*	No	Yes	No
Can you delete records and reuse space?	No***	Yes	Yes
Can you access the data set through an alternate index?	Yes	Yes	No
Can you REUSE the file?	Yes (if no AIX**)	Yes (if no AIX)	Yes
Can RBA or RRN change?	No	Yes	No

* You can insert records and change their lengths.

** RRN= relative-record number, RBA= relative-byte address, and AIX= alternative index.

*** You can, however, overlay a record if the length does not change.

VSAM Record Structure and Organization

Records in VSAM data sets are grouped into *control intervals*, the units 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 to 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 you (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 the 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. (See Figure 1.2 on page 6, Figure 1.3 on page 6, and Figure 1.4 on page 7 for depictions of the control interval formats used by each of the data set types.) KSDS control areas are used for distributing free space throughout the data set, as a percentage of control intervals per control area.

Figure 1.2 *ESDS Control Intervals and Control Areas*

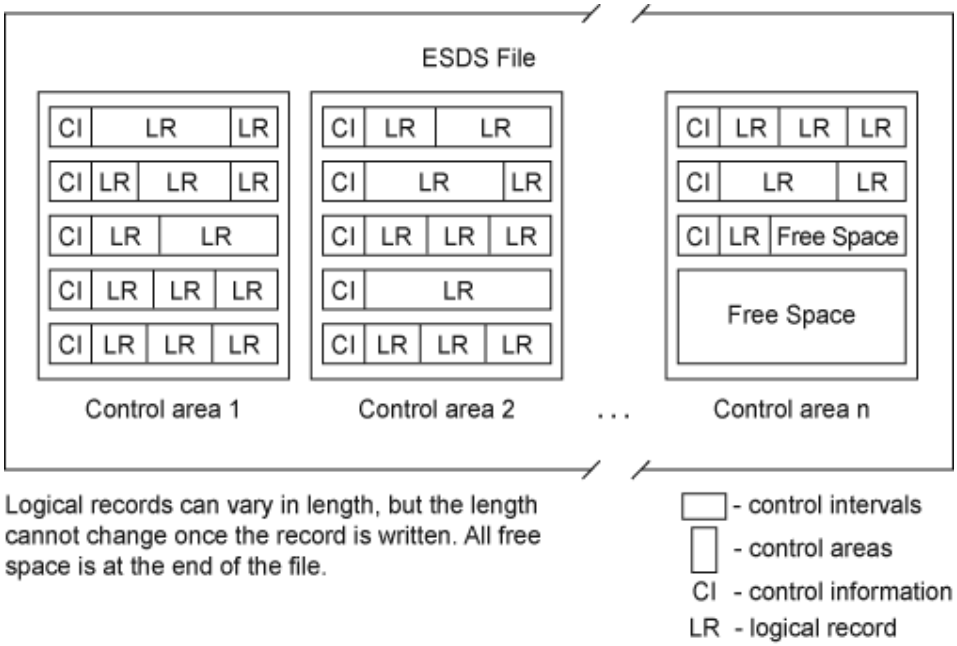


Figure 1.3 *KSDS Control Intervals and Control Areas*

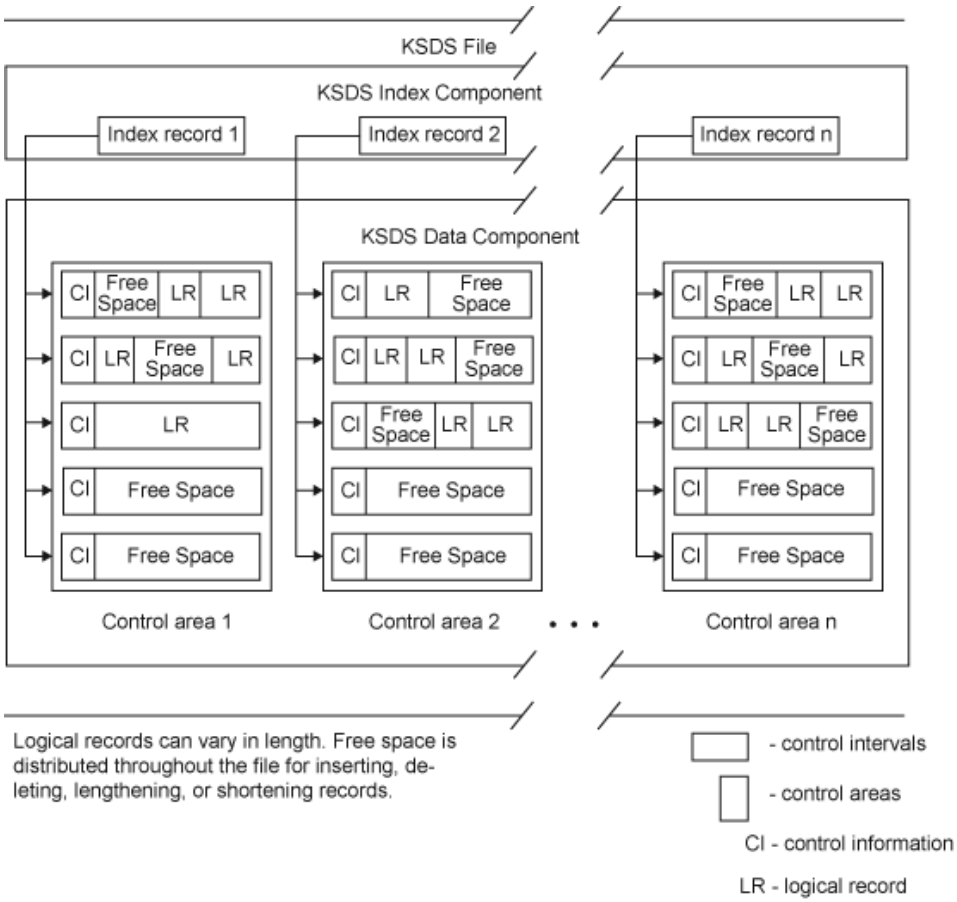
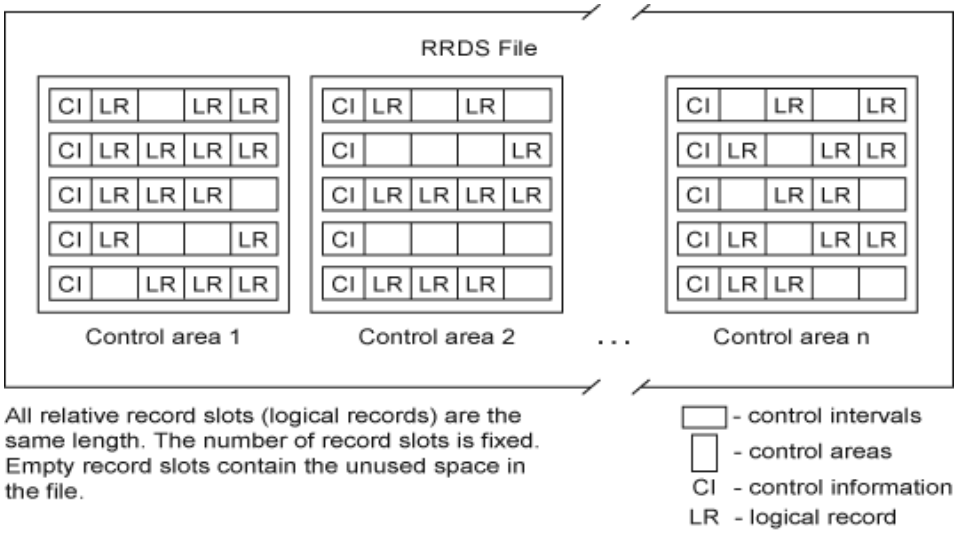


Figure 1.4 *RRDS Control Intervals and Control Areas*



VSAM Data Access Types

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

Introduction to 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 its relative-record number, or 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, there are two methods in which records are retrieved or stored:

- 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 that is wanted. The RRN is relative to the first record in the data set.

SAS 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

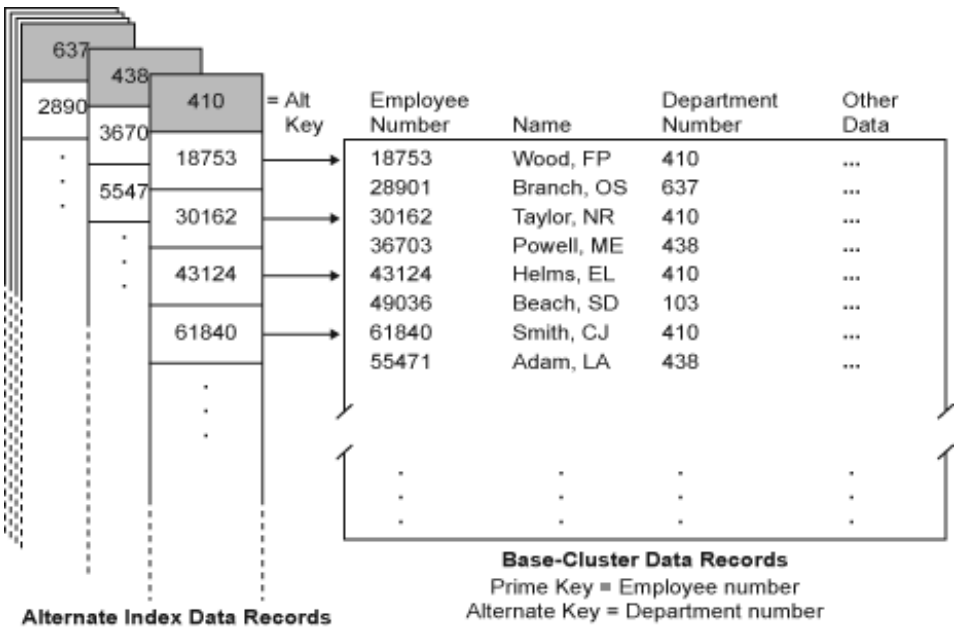
In addressed direct access, the entire data set is treated as a continuous stream of bytes. A record is retrieved and stored directly by its address relative to the beginning of the data set (relative-byte address, or RBA), which is dependent on the record's location relative to records previously accessed. SAS 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. 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.

The following figure 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.

Figure 1.5 *KSDS with Alternate Index (Nonunique Alternate Keys)*



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 or records 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 or records.

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. "Using Alternate Indexes for VSAM Data Sets" on page 93See and "IBM Documentation" on page 111 for more information about 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. The process uses keyed direct access to find a starting point. After 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 for the following reasons:

- Skip sequential processing can improve performance and reduce overhead, because a simple sequential retrieval is faster than direct retrieval.
- It enables you to find records when you know the key, RBA, or RRN of the first record that you want, but do not know the key, RBA, or RRN of the subsequent records.
- It enables you to process the data set sequentially, starting at a record other than the first.

Chapter 2

SAS Options and Variables for VSAM Processing

Introduction to SAS Options and Variables for VSAM Processing

SAS 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.

Global SAS system options and the automatic SAS variables are set when you process VSAM data sets. "Special SAS Options for VSAM" on page 15 presents a complete, descriptive list of each INFILE option available for processing your VSAM data sets.

See "Processing VSAM Data Sets in SAS Programs" on page 27 and the appropriate reference section for the VSAM data set that you want to access for information about using these options to process VSAM data.

Using SAS System Options

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

VSAMREAD
: enables or disables the reading of VSAM data sets.

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

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

VSAMRLS
enables or disables record-level sharing for a VSAM data set.

SAS is distributed with the following default options:

- VSAMREAD
- NOVSAMUPDATE
- NOVSAMLOAD
- VSAMRLS

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

Using 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.) The following automatic variables are not added to any SAS data sets created by a DATA step:

RBA
contains the RBA of the last record accessed.

Note: The _RBA_ variable is set to zero when a path is defined over an alternate index.

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:

BLKSIZE=*value*
specifies the block size of the input data set.

COLUMN=*variable*
defines a variable that SAS 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 SAS 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 a 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 SAS 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 SAS 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. The physical pathname 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 that is 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, SAS 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 SAS uses 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 SAS sets to the length of the current line.

LINE=*variable*
defines a variable, whose name you supply, that SAS 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 that was 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 that is specified by LINESIZE=, the action that is taken depends on which of the FLOWOVER, MISSOVER, and STOPOVER options is in effect. (By default, the MISSOVER option is in effect when VSAM data sets are read.)

If a PUT statement attempts to write a record longer than the value that is specified by LINESIZE=, the action that is taken depends on which of the FLOWOVER, MISSOVER, and STOPOVER options is in effect. (By default, the STOPOVER option is in effect when VSAM data sets are written.)

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 that you want available to the input pointer. The N= option cannot be used with direct access because, by definition, direct access gets only one line at a time.

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

PAD|NOPAD
specifies whether records that are 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 that is specified in the @'character string' expression (in 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, SAS sets _ERROR_ to 1, stops building the data set as if 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 SAS not to perform a buffered read.

For more information about standard and nonstandard SAS options, see the "INFILE Statement" in *SAS 9.3 Statements: Reference*.

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 about FILE statement options for VSAM data sets, see "Loading Records into a VSAM Data Set " on page 47.

All the options except those described in the following table can be used with all three VSAM data set types. The special SAS options for VSAM are listed in the following table:

Table 2.1 *Special SAS Options for Selected VSAM Data Sets*

Option	ESDS through AIX	ESDS	KSDS	RRDS
ERASE=			X	X
GENKEY	X		X	
KEY=	X		X	
KEYGE	X		X	
KEYLEN=	X		X	
KEYPOS=	X		X	
RBA=	X	X	X	
RRN=				X
SEQUENTIAL		X		X
SKIP	X		X	X

BACKWARD
BKWD
instructs SAS 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 SAS 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 SAS sets the FEEDBACK= variable, you must reset it to 0 to continue processing after a logical error. See “Error-Handling Techniques and Error Messages” on page 97 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. SAS 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 that are 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, SAS 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. Changing this variable 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. SAS 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 that is 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 that is 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 that are 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. 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).

SAS sets the variable that is specified by the KEYLEN= option to the actual key length that is 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 might 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 SAS sets to the position of the key field. This option enables 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 that is 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. Here are the guidelines for an appropriate password:

- You need a read (or higher-level) password for a data set that you are reading only.
- You need an update (or higher-level) password for a data set that you are updating or loading.
- You need 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.

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

RECORDS=*variable*
defines a numeric variable that SAS 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 or statements.

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 then 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 SAS that not every record that is 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

that are read need to be updated. You can set the UPDATE= variable to one of the following values:

- A value of 1 before an INPUT statement indicates that the record is retrieved for update. This is the default if UPDATE= is not specified.
- A value of 0 before the INPUT statement indicates that the record is not retrieved for update.

VSAM
indicates that the fileref points to a VSAM nonstandard, external data set. It is optional for the VSAM option to immediately follow the fileref in the INFILE and FILE statements. However, the VSAM option must immediately follow the fileref if 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 that are used for processing VSAM data sets fall into three functional categories:

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

See Table 6.1 on page 52 , Table 7.1 on page 62 , and Table 8.1 on page 80 for information about 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.

There are 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 "Defining and Loading a VSAM Data Set" on page 45 .

Here is the syntax for using the special options in the INFILE statement:

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

data-set-specification is a SAS fileref or a physical filename, 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

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. The following values are valid:

KS
specifies a VSAM key-sequenced data set.

ES
specifies a VSAM entry-sequenced data set.

RR
specifies a VSAM relative-record data set.

Chapter 3

Accessing VSAM Data Sets

Accessing VSAM Data Sets

Use the VSAM option to indicate that a fileref points to a VSAM external file.

Note: Many VSAM-specific options are available with the INFILE and FILE statements. For more information, see “VSAM Options for the FILE and INFILE Statements ” on page 23.

Reading a VSAM File

To read a VSAM file with an INPUT statement, specify the VSAM option in an INFILE statement:

```
filename in1 'prod.payroll';
data mydata;
   infile in1 vsam;
   input ...;
   /* SAS statements */
run;
```

Note: A VSAM file can be read sequentially without your having to specify the VSAM option.

Writing to an Empty VSAM File

To write to an empty VSAM file with a PUT statement, specify the VSAM option in a FILE statement:

```
filename out 'myid.newdata' disp=old;
data current;
   file out vsam;
   put ...;
   /* SAS statements */
run;
```

Updating a VSAM Data Set

To update a VSAM data set, include an INFILE statement and a FILE statement that point to the same fileref, and specify the VSAM type option in the DATA step:

```
filename mydata 'myid.newdata' disp=old;
data newdata;
   file mydata vsam;
   infile mydata vsam;
   /* SAS statements */
run;
```

Using Record-Level Sharing with VSAM

SAS provides support for the record-level sharing (RLS) access feature for VSAM data sets. For the RLS access feature to work, you must define your VSAM clusters as eligible for RLS access.

RLS eligible data sets must be SMS data sets that were defined with a LOG specification. The details of RLS definition, restrictions, and use are contained in the *IBM Data Facility Storage Management Subsystem (DFSMS)* documentation.

SAS determines whether a VSAM data set is RLS eligible when it opens the data set. If the data set is RLS eligible, SAS automatically opens it in RLS mode except in the following circumstances:

- A data set LOAD operation will be performed.
- The SMSVSAM server is not available.

  Note: If the SMSVSAM server is available but not accessible (thus causing an RLS OPEN to fail), SAS attempts to open the data set in non-RLS mode.

When a data set is opened in RLS mode, the following note is written to the SAS log:

```
NOTE: Fileref <fileref_name> opened in RLS mode.
```

The RLS default action can be overridden by specifying either VSAMRLS or NOVSAMRLS system options, which apply to all data sets referenced during the SAS

session. To override the RLS default action to a specific data set, you can use the RLS or NORLS options with the FILE and INFILE statements. (See "VSAMRLS System Option" on page 106.)

Opening the data set in non-RLS mode might generate the following results:

- If you are opening the data set for output, then the OPEN operation will fail if another application has the data set open. Alternatively, an attempt to subsequently open the data set by another application will fail while the data set is open in non-RLS output mode by SAS.
- If you are opening the data set for input, the OPEN operation will succeed, even though the data set is open by another application, as long as you specify SHAREOPTIONS(2) when you define the VSAM cluster.

Once a data set has been opened by VSAM in RLS mode, integrity is handled according to the *IBM Data Facility Storage Management Subsystem (DFSMS)* documentation. SAS software does not make or enforce these integrity rules.

Note: RLS processing can impose a significant overhead, and is not recommended when large quantities of VSAM data are to be read or written. Overhead is minimized when the RLSREAD specification is set to NRI. This is the default setting. A value for RLSREAD can be set via an INFILE statement, or via the RLS specification of a JCL DD statement or TSO ALLOC command.

The operation of RLS is essentially transparent to users. However, make sure you specify DISP=SHR in the statement that defines the VSAM file that you are opening.

Extended-Format VSAM Data Sets

SAS supports extended-format VSAM data sets. These data sets are managed with SMS, and they are defined as extended format with the data class DSNTYPE=EXT parameter and sub-parameters.

Extended-format data sets are the basis for many new VSAM functions, such as data striping, host data compression for key-sequenced data sets, system-managed buffering, and extended addressability for data sets greater than 4 gigabytes in size.

See the IBM DFSMS documentation for information about defining these functions to SMS and for any restrictions for using these functions.

VSAM Options for the FILE and INFILE Statements

You can use the following options for VSAM files in the FILE statement and in the INFILE statement. (Unless otherwise indicated, the option can be used in both.)

BACKWARD | BKWD
: causes SAS to read the VSAM data set backwards (INFILE only).

BUFND=*value*
: indicates how many data buffers to use for the VSAM data set.

BUFNI=*value*
: indicates how many index buffers to use for the VSAM data set.

CONTROLINTERVAL | CTLINTV | CNV
: indicates that you want to read physical VSAM control interval records rather than logical records. This option is typically used for diagnostic purposes (INFILE only).

ERASE=*variable*
: defines a numeric SAS variable that you must set to 1 when you want to erase a VSAM record (INFILE only).

FEEDBACK=*variable* | FDBK=*variable*
: defines a numeric variable that SAS sets to the VSAM logical error code. This option is similar to the _FDBK_ automatic variable. When SAS sets the FEEDBACK variable, you must reset it to 0 in order to continue.

GENKEY
: causes SAS to use the KEY= variable as the leading portion of a record's key. VSAM retrieves the first record whose key matches the generic key (INFILE only).

KEY=*variable* | KEY=*(variable1 variable2 . . .)*
: indicates that direct keyed access is being used to read records either from a KSDS or from an ESDS via an alternate index. Also, the variable contains the key value to be used in the retrieval of a record (input) or the writing of a record (output) (INFILE ONLY).

KEYGE
: is used in conjunction with the KEY= option. KEYGE indicates that when KEY= is used in a retrieval request, SAS retrieves any record whose key is equal to or greater than the specified key. This option is useful when the exact key is not known (INFILE only).

KEYLEN=*variable*
: specifies a numeric SAS variable that, when used with GENKEY, specifies the length of the key that is to be compared to the keys in the file.

KEYPOS=*variable*
: indicates the numeric variable that SAS sets to the position of the VSAM key field. This option enables you to read keys without knowing the key position in advance. This variable is set to the column number (starting from 1).

NRLS
: specifies not to use record-level sharing (RLS) to open an RLS-eligible data set (INFILE only).

PASSWD=*value*
: gives the appropriate password for a VSAM data set that has password protection.

RBA=*variable*
: specifies a numeric variable that you set to the relative byte address (RBA) of the data record that you want to read. The RBA= option indicates that addressed direct access is being used; it is appropriate for KSDS and ESDS. If you specify the CONTROLINTERVAL option, you can use the RBA= option to access control records in an RRDS (INFILE only).

RC4STOP
: stops the DATA step from executing if a return code greater than 4 is returned by the operating environment when the VSAM data set is opened.

RECORDS=*variable*
: defines a numeric variable that SAS sets to the number of logical records in a VSAM data set that has been opened for input.

RECORG=*record-organization*
specifies the organization of records in a new VSAM data set. Use this option only if SMS is active. The following is a list of valid values:

KS
specifies a VSAM key-sequenced data set.

ES
specifies a VSAM entry-sequenced data set.

RR
specifies a VSAM relative-record data set.

LS
specifies a VSAM linear-space data set.

RESET
indicates that the VSAM file is reset to empty (no records) when it is opened. This option applies only to loading a VSAM data set that has been marked REUSE. You cannot use this option if the data set contains an alternate index.

RLS | NORLS (alias NRLS)
specifies record-level sharing (RLS) to open an RLS-eligible data set.

The RLS and NORLS options override the NOVSAMRLS and VSAMRLS system options.

Note: If RLS is specified for a data set that is not RLS eligible, then the specification is ignored.

RLSREAD=NRI | CR | CRE
enables you to specify the level of read integrity required for an RLS-eligible data set (INFILE only). The RLSREAD= option overrides a similar specification made on a JCL DD statement or TSO ALLOC command.

NRI
provides a minimum amount of data integrity. Default is NRI.

CR
provides more data integrity via record-locking mechanisms. Use with a CICS transactional processing system.

CRE
provides more data integrity via record-locking mechanisms. Use with a DFSMStvs transactional processing system.

RRN=*variable*
defines a numeric variable that you set to the relative record number (RRN) of the record that you want to read or write. This option indicates that keyed direct access is being used; it is appropriate for RRDS only.

SEQUENTIAL
specifies sequential VSAM record retrieval when either the RBA= (for an ESDS) or the RRN= option (for an RRDS) is specified (INFILE only).

SKIP
indicates skip-sequential processing of VSAM files. Skip-sequential processing finds the first record whose value is the same as the value that is specified by the KEY= option; records are read sequentially from then on (INFILE only).

UPDATE=*variable*
defines a numeric SAS variable that indicates that not every record that it reads is to be updated. Use this option when you are updating records in a VSAM data set

(INFILE only). When an INFILE and a FILE statement reference the same VSAM data set, records are retrieved for update by default.

In most cases when you retrieve a record 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 avoids 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 the INPUT statement, the record is retrieved for update. This value is the default if UPDATE= is not specified. If you set UPDATE=0 before the INPUT statement, the record is not retrieved for update.

VSMDBUG=*nnnn*
specifies the number of times a message (that contains the filename, function requested, return code, and reason code) is written to the SAS log. A message is written each time there is a system request, such as GET, POINT, PUT, and so on until the number indicated is reached. *nnnn* can be 1 to 9999.

Chapter 4

Processing VSAM Data Sets in SAS Programs

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 following note:

```
Type=NONINDEXED
```

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

```
Type=INDEXED
```

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

```
Type=NUMBERED
```

Referring to VSAM Data Sets

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 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.

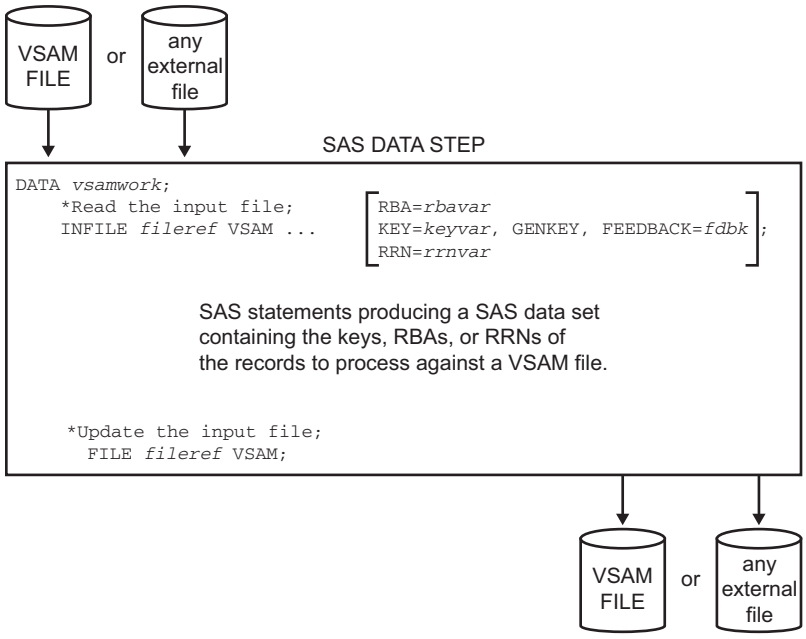
Operations on VSAM Data Sets in SAS Programs

SAS programs handle VSAM data sets the same as any external data set. The following are examples of operations that SAS can perform on *external data sets*, which are data sets that are not created by SAS:

- 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 from a VSAM data set. 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.

The following figure 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.) 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.

Figure 4.1 *Processing VSAM data Sets in SAS Programs*



You can perform five general types of operations on VSAM data sets in SAS programs:

- *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 "Defining and Loading a VSAM Data Set" on page 45 .

When you perform these operations, you can use certain types of access with each data set type. See the following table for an outline of this information. Note that VSAM provides both sequential and some form of direct access for each data set type.

Table 4.1 *Supported VSAM Operations and Access Types*

Operation	Access Type		ESDS	KSDS	RRDS
Read	Sequential:		Yes	Yes	Yes
	Direct access by:				
		Key	No	Yes	No
		Generic key	No	Yes	No
		RBA	Yes	Yes	No
		RRN	No	No	Yes
	Skip sequential:		No	Yes	Yes
Update	Sequential:		Yes	Yes	Yes
	Direct access by:				
		Key	No	Yes	No
		RBA	Yes	Yes	No
		RRN	No	No	Yes
Add or Load	Sequential:		Yes	Yes	Yes
	Direct access by:				
		Key	No	Yes	No
		RBA	No	No	No
		RRN	No	No	Yes
Erase	Sequential:		No	Yes	Yes
	Direct access by:				
		Key	No	Yes	No
		RBA	No	Yes	No
		RRN	No	No	Yes

Reading Records from a VSAM Data Set

Different Ways to Read Records

You must specify either the "VSAMREAD System Option" on page 105 or "VSAMUPDATE System Option" on page 107 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. "Processing an ESDS in a SAS Job" on page 51 , "Processing a KSDS in a SAS Job" on page 61 , and "Processing an RRDS in a SAS Job" on page 79 describe reading data from each of the three VSAM data set types.

Sequential Access

Sequential access means that a record is retrieved by one of the following methods:

- 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, SAS processes VSAM and other data sets sequentially. When access is sequential, SAS performs a standard look-ahead read. When SAS 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 using one of the following statements:

- the standard INFILE statement option EOF= and a STOP or SET statement
- a STOP statement to end the DATA step when you are processing the VSAM data set with direct access
- 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. After 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 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 do the following:

1. Specify the "VSAMUPDATE System Option" on page 107 .
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. Records are added in the following methods:

- 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

Introduction to Updating Records

Performing an update operation involves both input access (because the record must be read first) and output access (because you update by writing to the data set). Input access for an 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 use the following options and statements:

1. specify the "VSAMUPDATE System Option" on page 107 .
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 method enables 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 might 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 might 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 Records

To maintain data integrity in multiple update situations, 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 might 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 about using ENQ and DEQ, see "IBM Documentation" on page 111.

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 that is 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 "VSAMUPDATE System Option" on page 107. VSAM imposes a 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 SAS whether 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 might 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 that you might need for processing the data set in its INFILE statement.

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

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), SAS 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, you are reading from the data set before writing. SAS assumes that the data in the PUT statement is to update the record that you have just read with the INPUT statement unless you have changed one of the following:

- the RBA= variable value before the PUT statement executes for an ESDS. See "Combined Operations on an ESDS" on page 58 for more information.
- 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 74 for more information.
- the RRN= variable value before the PUT statement executes for an RRDS. See "Combined Operations on an RRDS" on page 89 for more information.

Examples of Using VSAM Data in SAS Programs

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 following example is based on the data set described in "Sample STUDENT Data Set" on page 109. The example can generate PROC PRINT listings:

```
  /* 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

The following example reads all of the records from the KSDS, 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. The following example is based on the data set described in "Sample STUDENT Data Set" on page 109.

```
   /* 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;
run;

proc sort data=test;
   by class;
run;

proc means data=test;
   var gpanum;
   by class;
run;

proc sort data=test;
   by state;
run;

proc means data=test;
   var gpanum;
   by state;
run;
```

Using a Windowing Program to Update VSAM Records

The following program is an example of using SAS Component Language (SCL) to create a simple windowing application for updating VSAM records.

This application uses SCL to provide an interface to a VSAM KSDS. The data set consists 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). You can also retrieve a record with a specific key by entering a Social Security number and selecting the RETRIEVE button. After 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 you select a button.
4. When you choose 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 the 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:

      /* 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.
         frstname $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 you want 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 z/OS 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                      */
```

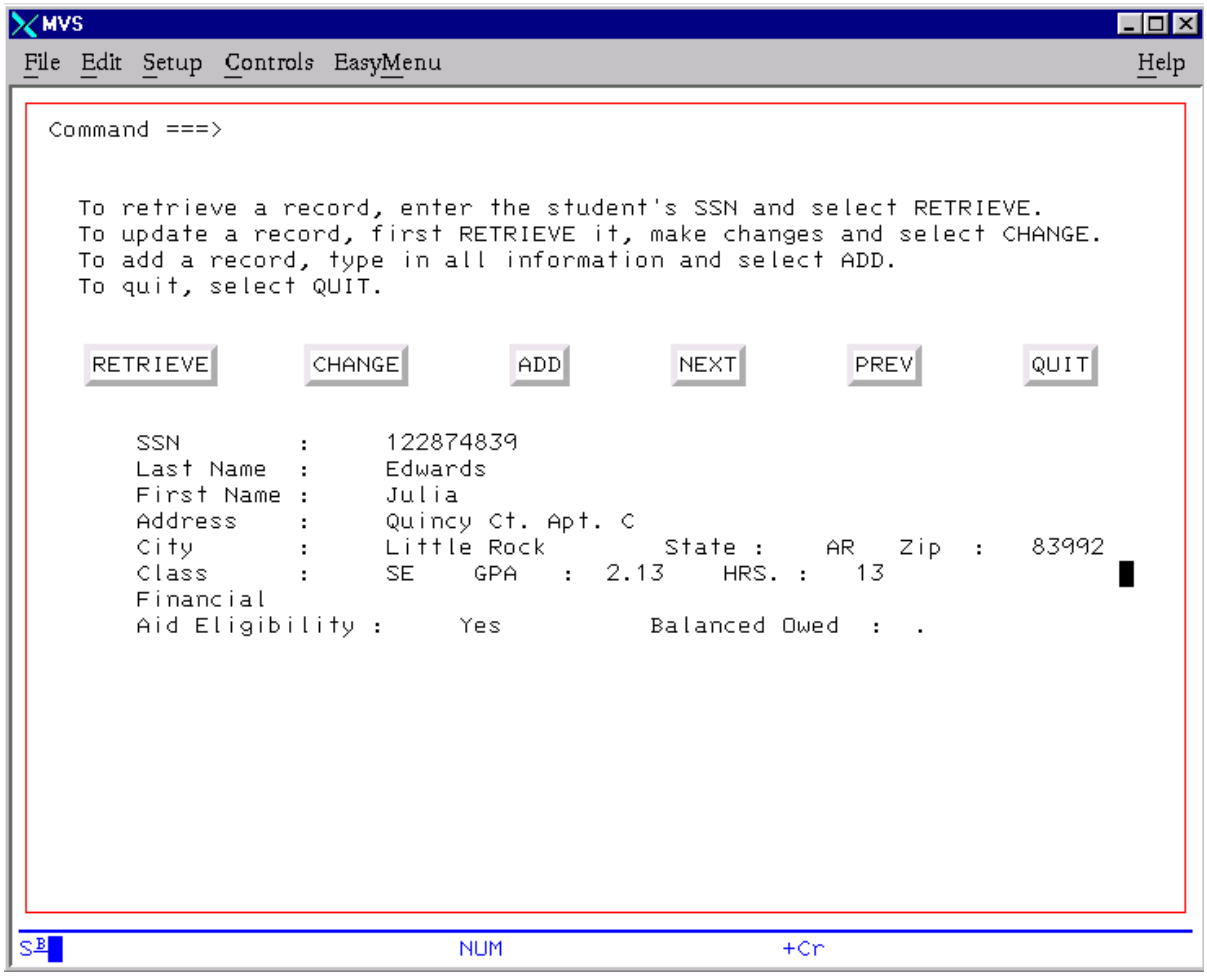
```
       @20 frstname $10.  /* Student's given name                 */
       @30 address  $25.  /* Permanent mailing address           */
       @55 city     $15.  /* City of residence                   */
       @70 state    $2.   /* State of residence                  */
       @72 zip      $5.   /* 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 explain  */
      /* 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 whether a record with that SSN already exists. If */
      /* so, explain. Else add a new observation to the    */
      /* data set and update its variables.                        */
   clause="id='"||idnum||"'";
   put clause=;
   rc=where(dsid,clause);
   put rc=;
   rc=fetchobs(dsid,1);
   put 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;
```

The output for the example creates the following window:

Display 4.1 *Result of SCL Windowing Program*



Chapter 5

Defining and Loading a VSAM Data Set

Defining a VSAM Data Set

You define a VSAM data set by using the IBM Access Method Services (AMS) IDCAMS utility, which is invoked from JCL. The following 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.

You can also define a VSAM data set in two other ways:

- by building a SAS macro variable
- by issuing a TSO DEFINE command in the Program Editor with the X statement

The "VSAMLOAD System Option" on page 105 and "VSAMUPDATE System Option" on page 107 are necessary for loading and updating VSAM data sets.

The following is an example of a macro variable:

```
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:

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

```
      CONTROLINTERVALSIZE(1024)
     )
;
```

Loading Records into a VSAM Data Set

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, SAS 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 "VSAMLOAD System Option" on page 105 . After the data set is loaded and closed, you can add, update, or erase records when the "VSAMUPDATE System Option" on page 107 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 that 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 be loaded only sequentially.

Options Used When Loading Records into a New VSAM Data Set

When you load initial records into a new VSAM data set, use only a FILE statement and specify the "VSAMLOAD System Option" on page 105 . In addition, you can use the following options in the FILE statement when you load a VSAM data set:

Table 5.1 *Options Used in the FILE Statement*

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

Access Types When Loading Records into a VSAM Data Set

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. After the data set is closed with one or more records, the load restrictions no longer apply. Loading the data set in this manner might 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 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 "VSAMLOAD System Option" on page 105 .
- 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 "Sample STUDENT Data Set" on page 109.

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

```
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 frstname $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;
```

Chapter 6

Processing an ESDS in a SAS Job

Introduction to ESDS

Record storage in an Entry-Sequenced Data Set (ESDS) 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.

The options associated with reading, adding, and updating ESDS records are described in the following sections. In many cases, the option's meaning depends on how it is used within a SAS program. (Loading ESDS records is discussed separately in "Defining and Loading a VSAM Data Set" on page 45.)

The following are three helpful tables and their descriptions:

- Table 6.1 on page 52 lists the options that are significant for processing ESDS records.
- Table 6.2 on page 53 summarizes the access type that you can use for each ESDS operation.
- Table 6.3 on page 55 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 data set described in "Sample STUDENT Data Set" on page 109.

For information about using a KSDS in SAS jobs, see "Processing a KSDS in a SAS Job" on page 61. For information about using an RRDS in SAS jobs, see "Processing an RRDS in a SAS Job" on page 79.

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 6.1 *SAS Options for an ESDS*

	Functional Category		
Option	**Informational**	**Record Retrieval**	**Record Storage**
BACKWARD		X	
BUFND=	X		
BUFNI=	X		
CONTROLINTERVAL		X	
ERRORABEND	X		
FEEDBACK=	X		
PASSWD=	X		
RBA=		X	
RECORDS=	X		
RESET			X
SEQUENTIAL		X	
UPDATE=		X	
VSAM	X	X	X

Reading Records from an ESDS

Access Types for ESDS Operations

You can use sequential or addressed direct access to read records from an ESDS within a SAS program. If the ESDS has an alternate key index, you can also use keyed direct access. (See “Keyed Direct Access by Alternate Keys” on page 54.) The options that are specified in the INFILE statement determine the access type for an ESDS read operation. Either the “VSAMLOAD System Option” on page 105 or the“VSAMUPDATE System Option” on page 107 must be specified in order to read VSAM data sets.

Table 6.2 *Access Types for ESDS Operations*

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

* 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:

```
/* 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

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 might 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 that 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 program illustrates addressed direct access to an ESDS:

```
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 instructions on how to create an alternate index for an ESDS, see “Using Alternate Indexes for VSAM Data Sets” on page 93 .

You can treat an ESDS accessed through an alternate index as if it were a KSDS, except that records cannot be erased and the record length cannot be changed.

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

Table 6.3 *Operations on an ESDS with an Alternate Index*

Operation	Access
Read	Sequential
	Direct by: • alternate key • RBA
Add	Sequential
Update	Sequential
	Direct by: • alternate key, if required • RBA
Load	Sequential

Adding Records to an ESDS

Add records to an existing ESDS using the following method:

1. Specify the "VSAMUPDATE System Option" on page 107 .
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, see Table 6.1 on page 52.

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. Here is an example:

```
data three;
infile myesds vsam ;
file myesds vsam ;
id='289478363';
lastname='Cox          ';
frstname='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 frstname $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       */
   newstu=newstu+1;
   retain newstu;
   ...more SAS statements...
run;
```

In the example, a record for a new student, JUNE COX, is defined and added to the MYESDS data set without first reading the ESDS. The record is added to the end of the data set because output access is always sequential when new records are added to an ESDS. (See “Access Types for ESDS Operations” on page 53.)

Updating Records in an ESDS

Steps for Updating Records in an ESDS

To update records in an ESDS, follow these steps:

1. Specify the “VSAMUPDATE System Option” on page 107 .
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, see Table 6.1 on page 52.

When records in an ESDS are updated, you have the following input and output access:

- 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 “Access Types for ESDS Operations” on page 53.)
- Output access for writing is direct because the last record that is read is the record updated.

Using the PUT Statement When Updating Records in an ESDS

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. 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;
```

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 33 for more information.

In the following example, 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 frstname;
   end;
run;
```

Combined Operations on an ESDS

You might 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 might need to process the ESDS in its INFILE statement.

In a DATA step that combines operations, SAS 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, SAS 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, SAS assumes that the data in the PUT statement is to modify the record that 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.)

```
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 7

Processing a KSDS in a SAS Job

Introduction to KSDS

Each record in a Key-Sequenced Data Set (KSDS) has a key that contains a unique value. 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. 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.

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 the following sections. In many cases, the option's meaning depends on how it is used within a SAS program. (Loading KSDS records is discussed separately in "Defining and Loading a VSAM Data Set" on page 45.)

Here are two helpful tables and their descriptions:

- Table 7.1 on page 62 lists the options that are significant for processing KSDS records.
- Table 7.2 on page 64 summarizes the access type for KSDS operations.

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 "Sample STUDENT Data Set" on page 109. You can run the examples by using the sample programs provided in the Help system and following the steps to define and load a KSDS described in "Defining and Loading a VSAM Data Set" on page 45 .

For information about using an ESDS in SAS jobs, see "Processing an ESDS in a SAS Job" on page 51 . For information about using an RRDS, see "Processing an RRDS in a SAS Job" on page 79 .

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 7.1 *SAS Options for a KSDS*

	Functional Category		
Option	**Informational**	**Record Retrieval**	**Record Storage**
BACKWARD		X	
BUFND=	X		
BUFNI=	X		
CONTROLINTERVAL		X	
ERASE=			X
ERRORABEND	X		
FEEDBACK=	X		
GENKEY		X	
KEY		X	X
KEYGE		X	
KEYLEN=	X		
KEYPOS=	X		
PASSWD=	X		
RBA=		X	
RECORDS=	X		

		Functional Category	
Option	**Informational**	**Record Retrieval**	**Record Storage**
RESET			X
SKIP		X	
UPDATE=		X	
VSAM	X	X	X

Reading Records from a KSDS

Three Ways for 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 7.2 on page 64 for more information.) 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:

```
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

Introduction to Reading a KSDS with Direct Access

A KSDS is read directly using the following methods:

- 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 that you want SAS 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 "SAS Options and Variables for VSAM Processing" on page 11 are used to retrieve KSDS records. These options are GENKEY, KEY=, KEYGE, KEYLEN=, and KEYPOS=.

Table 7.2 *Access Types for KSDS Operations*

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

* 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:

```
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 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 that is defined when the KSDS was created. (For example, if the KSDS 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 option 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:

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

If necessary, the value of KEYVAR is 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. SAS 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:

```
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, SAS 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 68 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, SAS sets the variable that is specified by KEYLEN= to the actual (maximum) key length that is defined in the KSDS data set. This option enables 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 might need to name the variable in a RETAIN statement if you need this initial value after the first execution of the DATA step:

```
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. 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 enables you to read KSDS keys without knowing their positions in advance.

```
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. See "Keyed Direct Access with an Alternate Index" on page 8 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 "Using Alternate Indexes for VSAM Data Sets" on page 93 for examples of the control language that defines an alternate index over a KSDS.

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:

```
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.) After 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 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 procedure output 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 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.

```
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 procedure output file, and go on to the next */
      /* observation.                                          */
   else do;
      sasrc =0;
      put 'There is no record with this generic key: ' keyvar;
   return;
   end;
 end;

      /* 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 procedure output file, and go on to the next */
         /* observation.                                          */

      else do;
         sasrc=0;
         put 'There is no record with this generic key: ' keyvar;
      return;
      end;
    end;
    input @ kp keyword1 $;
  end;
run;
```

Adding Records to a KSDS

To add records to a KSDS:

1. specify the “VSAMUPDATE System Option” on page 107.
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, see “Special SAS Options Used with a KSDS” on page 62 .

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.

```
data ten;
   infile myksds vsam;
   file myksds vsam;
   id='963215683';
   lastname='Flintstone  ';
   frstname='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 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.
       @90 finaid   $1.;
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 7.2 on page 64 for more information.) 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, complete the following steps:

1. specify the "VSAMUPDATE System Option" on page 107.
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, see "Special SAS Options Used with a KSDS" on page 62 .

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 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 might 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 myksds vsam keypos=kp;
   input;
   file myksds vsam;
   put @1 data1 @30 data2 @ kp keydata;
   …more SAS statements…
```

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 33 for more information.) The following program 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.

```
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;
```

Erasing Records from a KSDS

To erase a record from a KSDS, complete the following steps:

1. specify the "VSAMUPDATE System Option" on page 107 .
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 SAS whether a record is to be erased.

See Table 7.1 on page 62 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.

```
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

Introduction to Combined Operations on a KSDS

You might 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.

Adding Records without Reading

Introduction to 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), SAS 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 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 "Error-Handling Techniques and Error Messages" on page 97 for more information.)

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

- 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.

Here is an example:

```
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.
             @90 finaid   $1.;

      end;  /* If FDBK=8 */
      stop;
run;
```

Adding Records after Reading

Introduction to Adding Records after Reading

When you are reading from the KSDS before you write, SAS 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 a record with a particular key exists. You then 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 "Error-Handling Techniques and Error Messages" on page 97 for more information.)

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

- 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.

Here is an 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: you 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 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.
       @90 finaid   $1.;

   stop;
run;
```

Comparing Key-Testing Techniques

Notice the differences between the two key-testing techniques:

- 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 might 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.

Chapter 8

Processing an RRDS in a SAS Job

Introduction to Processing an RRDS

A Relative-Record Data Set (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.

You can read, add, update, and erase RRDS records in SAS programs. In many cases, the option's meaning depends on how it is used within a SAS program. (Loading RRDS records is discussed separately in “Defining and Loading a VSAM Data Set” on page 45 .)

Here are two helpful tables and their descriptions:

- Table 8.1 on page 80 lists the options that are significant for processing RRDS records.
- Table 8.2 on page 83 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 "Sample STUDENT Data Set" on page 109. You can run the examples by using the sample programs that are provided in the Help system and by following the steps to define and load an RRDS that are described in "Defining and Loading a VSAM Data Set" on page 45.

For information about using an ESDS in SAS jobs, see "Processing an ESDS in a SAS Job" on page 51. For information about using a KSDS in SAS jobs, see "Processing a KSDS in a SAS Job" on page 61.

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 8.1 *SAS Options for an RRDS*

	Functional Category		
Option	**Informational**	**Record Retrieval**	**Record Storage**
BACKWARD		X	
BUFND=	X		
BUFNI=	X		
CONTROLINTERVAL		X	
ERASE=			X
ERRORABEND	X		
FEEDBACK=	X		
PASSWD=	X		
RRN=		X	X
RECORDS=	X		
RESET			X
SEQUENTIAL		X*	
SKIP		X	
UPDATE=		X	

	Functional Category		
Option	Informational	Record Retrieval	Record Storage
VSAM	X	X	X

* Meaningful only if you also have a PUT statement.

Reading Records from an RRDS

Three Ways of 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 8.2 on page 83). The type of RRDS Read operation is specified with the appropriate options in the SAS INFILE statement. You must specify either the "VSAMREAD System Option" on page 105 or the "VSAMUPDATE System Option" on page 107 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.

```
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. You might know the RRN of a record because it has some relationship to the record contents or because you have obtained it in some other way.

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 SAS to read. The variable is created if it does not exist and is not added to the output data set.

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. After 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 might be empty.

When you process skip sequentially, you must specify a means of stopping the DATA step. In the following example, 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 procedure output 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:

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

data process;
   set rrnums;
   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;
      return;
      end;
   end;

         /* Read next records sequentially while the class matches. */

         /* Write the records to the procedure output 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 8.2 *Access Types for RRDS Operations*

Operation	Read (INFILE/INPUT Statements)	Write (FILE/PUT Statements)
Read	Sequential	Does not apply
	Direct with RRN= option	
	Skip sequential with SKIP and RRN= options	

Operation	Read (INFILE/INPUT Statements)	Write (FILE/PUT Statements)
Add*	Direct with RRN= option	Must be direct: use the RRN= option
	Sequential with SEQUENTIAL and RRN= options	
Update	Sequential	Direct: the record read is the record updated
	Direct with RRN= option	
Erase	Sequential	Direct: the record read is the record erased
	Direct with RRN= option	
Load	Does not apply	Sequential: in relative-record order
		Direct with RRN= option

* The INPUT statement is not required.

Adding Records to an RRDS

Introduction to Adding Records to an RRDS

To add records to an RRDS, complete the following steps:

1. Specify the "VSAMUPDATE System Option" on page 107 .
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 8.2 on page 83 .

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 8.2 on page 83). 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.

```
data addrrns;
      /* Create a new RRDS record and assign it to the */
      /* NEWREC variable.                              */
   length newrec $90.;
   id='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';
   newrec = id||lastname||frstname||address||city||state||
            zip||balance||gpa||class||hrs||finaid;

      /* Assign the RRN of the new record (NEWREC) to */
      /* the ADDRRN variable.                         */
   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 record that 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. 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.

```
data addrrns;

     /* Create a new RRDS record and assign it to the */
     /* NEWREC variable.                              */
   length newrec $90.;
   id='995613769';
   lastname='Rubble    ';
   frstname='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||frstname||address||city||state||
            zip||balance||gpa||class||hrs||finaid;
      /* Assign the RRN of the new record (NEWREC) to   */
      /* the ADDRRN variable.                           */
   addrrn=32;
run;

data readrrns;
   input readrrn ;
   datalines;
31
;
run;

data five;
   set readrrns;
   set addrrns;
   rrnvar=readrrn;

      /* Read the RRDS record specified by the READRRN variable. */
   infile myrrds vsam rrn=rrnvar;
   file myrrds vsam ;
   input;
```

```
      /* Add the new RRDS record with direct access by assigning the */
      /* value of ADDRRN to the RRNVAR variable and writing the      */
      /* NEWREC variable that contains the complete new record.      */
   if (readrrn = 31) then do;
      put _infile_;
      rrnvar =addrrn;
      put newrec;
   end;
run;
```

Updating Records in an RRDS

To update records in an RRDS, complete the following steps:

1. Specify the "VSAMUPDATE System Option" on page 107.
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, see Table 8.1 on page 80.

When you update a record in an RRDS, input access for reading can be either sequential or direct. (For more information, see Table 8.2 on page 83). 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. For more information, see "Reading an RRDS with Skip Sequential Access" on page 82 .

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 might 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. 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;
```

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.

For more information, see "Using the UPDATE= Option" on page 33 . 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.

```
data rrnumbrs;

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

data six;
   set rrnumbrs;
   updtevar =0;
   infile myrrds vsam rrn=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, complete the following steps:

1. Specify the "VSAMUPDATE System Option" on page 107.
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 SAS whether a record is to be erased.

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.

```
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

How to Combine 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.

SAS determines whether you want to add new or update existing RRDS records.

Adding Records without Reading

Adding Records without Reading Overview

When you do *not* execute an INPUT statement before the PUT statement (because you are adding records without reading from the RRDS), SAS 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 "Error-Handling Techniques and Error Messages" on page 97 for more information.)

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

- 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.

```
data rrdsinfo;
      /* Select values for lastname,firstname, and class. */
   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 rrdsinfo;
   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  */
      /* already exists, read it in and update it.            */
   if fdbk=8 then do;
       fdbk=0;
       _error_=0;
       input;
       put @1 _infile_ @86 class $2.
   end;
run;
```

Adding Records after Reading

Adding Records after Reading Overview

When you read from the RRDS before you write, SAS 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. The FEEDBACK= option specifies a SAS variable that is set to the VSAM logical error code when a logical error occurs. (See "Error-Handling Techniques and Error Messages" on page 97 for more information.)

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

- 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.

```
data rrdsinfo;
     /* Select values for lastname, firstname, and class. */
   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 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 frstname  $10.
          @86 class     $2.;

         /* 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 frstname  $char10.
              @86 class     $char2.;
      end;
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:

- 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 might 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 9

Using Alternate Indexes for VSAM Data Sets

Introduction to Using Alternate Indexes

An alternate index provides another way to access a VSAM data set, as described in "Keyed Direct Access with an Alternate Index" on page 8. You define and build alternate indexes by using IBM Access Method Services (AMS).

You can create an alternate index for a new data set or an existing data set, and for an ESDS or a KSDS. An alternate index is itself a KSDS that references the base cluster.

In the following examples, the alternate index enables you to perform direct keyed access of the STUDENT data set by state code. See "Sample STUDENT Data Set" on page 109 for the data set that is used in the examples.

Creating an Alternate Index for an ESDS

You can create an alternate index for an ESDS.

The following example shows the IBM AMS IDCAMS job that is needed to create an alternate index for an ESDS. An IDCAMS job can be submitted using 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 94 , 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) );
%let def=&def %str(volumes(xxxxxx) );
%let def=&def %str(reuse);

   /* Specify NONUNIQUEKEY because there is more than one record */
   /* with the same state code.                                  */
%let def=&def %str(nonuniquekey );
%let def=&def %str(records (10 5) );
%let def=&def %str(cisz(2048) );

   /* Use the two-letter state code that is offset 69 bytes      */
   /* in each record.                                            */
%let def=&def %str(keys(2 69)%) );

   /* Define a path as a reference to the alternate index.       */
%let path=%str(define path %(name('dsname.esds.student.path') );
%let path=&path %str(pathentry('dsname.esds.myindex')%) );

   /* Build the index from data contained in INDATASET,          */
   /* and put it into the index specified in OUTDATASET.         */
%let bld = %str(bldindex indataset('dsname.esds.student') );

%let bld=&bld %str(outdataset('dsname.esds.myindex') );

%sysexec &def;
%sysexec &path;
%sysexec &bld;

   /* Assign a fileref to the index path.                        */
filename mypath 'dsname.esds.student.path ' disp=shr;

   /* Read the data from the data set 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;

   /* It is a good practice to clear filerefs when you are       */
   /* done with them, but it is not necessary.                   */
filename mypath  clear;
```

Creating an Alternate Index for an Existing KSDS

You can create an alternate index over an existing KSDS by using IDCAMS using 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, SAS is invoked to read the data set using the alternate index and to write the records to the procedure output file.

```
//DALTINDX JOB accounting information
//*
//*   Define an alternate key for an existing KSDS.
//*
//STEP1    EXEC  PGM=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 SAS 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 KSDS via the alternate key.  Write the records */
   /* to the procedure output 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 SAS 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 CONTROLINTERVALSIZE. Use the following formula to calculate the maximum record size of spanned records:

- MAXLRECL=CI/CA x (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 ESDS or KSDS.

- ESDS:
  - RECSZ= 5 + AIXKL + (*n* x 4)
- KSDS:
  - RECSZ= 5 + AIXKL + (*n* x BCKL)
- 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 this 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, the average record size for alternate key STATE was calculated as follows:

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

Specifying the same value for average and maximum identifies the records as fixed length. See “IBM Documentation” on page 111 for more information about calculating record size.

Chapter 10

Error-Handling Techniques and Error Messages

What are Physical and Logical Errors?

Two types of errors can occur when you use SAS to process VSAM data sets. The types of errors are physical and logical. There are several things that you need to know:

- the differences between physical and logical errors
- error detection
- error-handling techniques
- the FEEDBACK= variable and other error detection variables
- VSAM feedback codes
- how COBOL status key values correspond to VSAM feedback codes (see "COBOL Status Key Values and VSAM Feedback Codes" on page 103).

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, SAS does the following:

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

Types of Logical Errors

Logical 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 SAS 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 SAS detects an error condition, it is a SAS logical error. When SAS 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 enables 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 the following:

- partially built records to be written to the data set
- the DATA step to terminate immediately with an error message

For more information about these options, see "Processing VSAM Data Sets in SAS Programs" on page 27.

VSAM Logical Errors

Errors in your program logic that VSAM detects are called VSAM logical errors. Here are some common VSAM logical errors:

- 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. The _FDBK_ variable 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. However, the most common values are the same for all operating systems. For 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 type of logical error was detected. "Some Common Causes of Logical Errors" on page 101 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 might occur. The ability to reset the FEEDBACK= variable after taking appropriate action to handle the error is very significant. For this reason, it is strongly recommended that you use the FEEDBACK= option for all VSAM data sets in which logical errors might occur.

Other distinctions are that _FDBK_ is also set if the following occurs:

- when VSAM detects a physical error.

- when VSAM sets a zero return code in certain situations.

  You get a nonzero _FDBK_ with a zero _IORC_ 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

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 enables 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, SAS 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 "Processing a KSDS in a SAS Job" on page 61) and the slot-testing techniques for an RRDS (described in "Processing an RRDS in a SAS Job" on page 79).

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:

```
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 might want to print error messages warning you 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 might also design logic to handle specific, anticipated FEEDBACK= variable values.

Some Common Causes of Logical Errors

The error condition that is associated with each feedback code is briefly described in the list of the VSAM feedback codes. The codes in this list represent decimal values.

IBM documentation describes 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 SAS detects an error before it invokes VSAM. Instead, SAS 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 following occurs:

- SAS cannot detect the error in advance (for example, user lockout).
- SAS 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 10.1 *VSAM Feedback Codes and Error Descriptions*

Feedback Code	Error Description
4	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.
8	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.)
12	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).
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.
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 that 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, SAS assumes that you intend for the record to be a new record (if the new key is not a duplicate).

Feedback Code	Error Description
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 completed the following: • copied the input record to the PUT buffer with a PUT _INFILE_ @ statement • built the key in the PUT buffer when creating a new record • 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

The COBOL status key values correspond to the VSAM feedback codes. Before you invoke VSAM, SAS 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 10.2 *COBOL Status Key Values and VSAM Feedback Codes*

VSAM Feedback Code	COBOL Status Key Value
8	22
12	21
16	23
20	93
32	90
36	92
88	21
96	94
108	92

VSAM Feedback Code	COBOL Status Key Value
192	23

For more information about COBOL status key values and their corresponding VSAM feedback codes, consult "IBM Documentation" on page 111.

Appendix 1

VSAM System Option Dictionary

Dictionary

VSAMLOAD System Option

Enables you to load a VSAM data set.

Valid in: Configuration file, SAS invocation, OPTIONS statement, OPTIONS window

Category: File Control: EXTFILES

PROC OPTIONS GROUP= EXTFILES

Default: NOVSAMLOAD

z/OS specifics: All

Syntax

VSAMLOAD | NOVSAMLOAD

Required Argument

VSAMLOAD
must be in effect in order to load an empty VSAM data set. RLS data sets are always loaded in non-RLS mode.

VSAMREAD System Option

Enables the user to read a VSAM data set.

Valid in: Configuration file, SAS invocation, OPTIONS statement, OPTIONS window

Category: File Control: EXTFILES

PROC OPTIONS GROUP= EXTFILES

Default: VSAMREAD

z/OS specifics: All

Syntax

VSAMREAD | NOVSAMREAD

Optional Argument

VSAMREAD
enables you to process VSAM data sets with a SAS DATA step.

VSAMRLS System Option

Enables record-level sharing for a VSAM data set.

Valid in: Configuration file, SAS invocation, OPTIONS statement, OPTIONS window

Category: Files: EXTFILES

PROC OPTIONS GROUP= EXTFILES

Alias: RLS | NORLS

Default: VSAMRLS

z/OS specifics: All

Syntax

VSAMRLS | NOVSAMRLS

Optional Arguments

VSAMRLS
specifies that record-level sharing is supported.

NOVSAMRLS
specifies that SAS is not to attempt to open a VSAM data set in record-level sharing mode, even if the data set is defined as VSAMRLS eligible.

VSAMRLSREAD System Option

Specifies the level of read integrity for an RLS-eligible data set.

Valid in: Configuration file, SAS invocation, OPTIONS statement, OPTIONS window

Category: File Control: EXTFILES

PROC OPTIONS GROUP= EXTFILES

Default: VSAMRLSREAD

z/OS specifics: All

Syntax

VSAMRLSREAD | NOVSAMRLSREAD

Optional Argument

VSAMRLSREAD
enables you to specify the level of read integrity required for an RLS-eligible data set.

VSAMUPDATE System Option

Enables you to update a VSAM data set.

Valid in: Configuration file, SAS invocation, OPTIONS statement, OPTIONS window

Category: File Control: EXTFILES

PROC OPTIONS GROUP= EXTFILES

Default: NOVSAMUPDATE

z/OS specifics: All

Syntax

VSAMUPDATE | NOVSAMUPDATE

Required Arguments

VSAMUPDATE
must be in effect in order to update VSAM data sets.

VSAMUPDATE
implies VSAMREAD.

Appendix 2

Sample STUDENT Data Set

The examples use the fictional STUDENT data set that is shown in this section. 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 z/OS. You can then enter or copy the code for the other examples. The sample programs are available in the Help system and on the installation media.

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

OBS	ID	LASTNAME	FRSTNAME	ADDRESS	CITY
1	122874839	Edwards	Julia	2450 Quincy Ct. Apt. C	Little Rock
2	145637205	Martin	Duanne	392 Hazelwood Dr.	New Hartford
3	167294367	Smith	Jerry	111 Lincoln Ave.	Boston
4	194304428	Allen	Nancy	423 Lakefront Dr.	Deerborne
5	234355167	Teague	Denise	556 Cherokee Rd.	Oklahoma City
6	237849217	Jones	Antony	110 Aberdeen Rd.	Albany
7	274596043	Friedman	Oscar	2845 Ocean Drive	Tampa
8	289478363	Cox	June	Rt. 2 Box 784	Cheyenne
9	293652329	Hawthorne	Jean	688 Ridge Rock Way	Bountiful
10	357593476	Doe	John	384 Main Street	Walla Walla

OBS	ID	LASTNAME	FRSTNAME	ADDRESS	CITY
11	367829047	Taylor	Anne	47 Lawrence Circle	Lawrence
12	372054321	Hall	Brad	56 Starr Ave.	Phoenix
13	378462917	Starnes	Randy	1450 Rock Quarry Road	Lexington
14	467879765	Mitchell	Barbara	923 Kemper Court	Spartanburg
15	478369204	Ward	Keith	Box 2330 Hwy 90	Kalamazoo
16	483029412	Henson	Edward	783 12th Ave. Circle	Knoxville
17	547293675	Pierce	Timothy	1233 Hamilton Drive	Dallas
18	547392749	Thomas	Matthew	Rt. 4 Box 634	Reading
19	567879343	Thomas	Wanda	21 Martian Way	Jupiter
20	578927349	Miller	Frank	570 8th Avenue	New York

OBS	ID	LASTNAME	FRSTNAME	ADDRESS	CITY
21	638798462	Jones	Tanya	289 Jones Street	San Diego
22	648309214	Bradley	Steve	PO Box 282	Albuquerque
23	674930930	Olsen	Wayne	PO Box 2580	Chicago
24	703946238	Allen	Beth	2834 Harcourt St.	Seattle
25	743092873	Miller	Caroll	Rt. 3 Box 245	Jackson
26	804763829	Thomas	Henry	397 Pennsylvania Ave.	Washington
27	847204826	Allen	Harold	56 48th Street	Denver
28	867496732	Quimbley	Fred	934 Oak Street	Richmond
29	904873627	Hart	Jim	489 Hartford Drive	Miami
30	948372958	Hill	Thomas	2458 Johnson Parkway	Santa Monica

OBS	STATE	ZIP	BALANCE	GPA	CLASS	HRS	FINAID
1	AR	83992	00050	2.13	SE	13	Y
2	CN	00103	00000	0.75	JU	13	Y
3	MA	00376	00025	4.00	FR	00	Y
4	MI	50471	00025	2.45	GR	07	N
5	OK	53062	00000	2.63	SO	20	Y
6	NY	10025	00000	2.35	FR	05	Y
7	FL	34988	00035	1.22	FR	09	Y
8	WY	59334	00100	2.33	SE	13	Y
9	UT	79483	00050	2.75	SO	00	Y
10	WA	90126	00025	1.55	GR	00	Y

OBS	STATE	ZIP	BALANCE	GPA	CLASS	HRS	FINAID
11	KS	69037	00000	3.22	JU	13	Y
12	AZ	82453	00000	3.46	SO	06	Y
13	KY	48223	00000	2.89	SO	14	N
14	SC	29027	00050	2.98	FR	00	Y
15	MI	47893	00000	2.48	SO	03	Y
16	TN	37294	00150	2.93	GR	14	N
17	TX	55934	00000	0.43	SE	00	Y
18	PA	16382	00025	3.33	JU	12	N
19	FL	34892	00000	1.83	SE	00	N
20	NY	10003	00100	1.95	GR	08	N

OBS	STATE	ZIP	BALANCE	GPA	CLASS	HRS	FINAID
21	CA	97274	00000	4.00	SE	14	Y
22	NM	73485	00020	3.80	SO	17	N
23	IL	56038	00000	3.85	SE	10	Y
24	WA	91673	00000	2.37	FR	17	N
25	MS	42865	00025	4.00	JU	12	Y
26	DC	19534	00000	2.50	GR	10	Y
27	CO	60049	00050	1.83	JU	16	N
28	VA	20784	00070	3.90	JU	00	Y
29	FL	33134	00000	3.55	GR	11	Y
30	CA	99999	00000	4.00	GR	11	Y

Appendix 3

IBM Documentation

The following list summarizes references for IBM documentation.

- *Device Support Facilities User's Guide and Reference (GC35–0033)*
- *MVS/ESA Integrated Catalog Administration: Access Method Services Reference (SC26–4500)*
- *MVS/XA Integrated Catalog Administration: Access Method Services Reference (GC26–4135)*

Glossary

addressed direct access
a method of access in which 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
See 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 to build an alternate index.

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 data component, an index component, or both.

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.

DATA step view
a type of SAS data set that consists of a stored DATA step program. A DATA step view contains a definition of data that is stored elsewhere; the view does not contain the physical data. The view's input data can come from one or more sources, including external files and other SAS data sets. Because a DATA step view only reads (opens for input) other files, you cannot update the view's underlying data.

Entry-Sequenced Data Set
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
See Entry-Sequenced Data Set.

key
a value that uniquely identifies a specific record and its order among other records in a database.

key field
See sequence field.

keyed direct access
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.

logical record
data that is requested of or given to the data management function (VSAM in this case) as a unit.

path
the route through a hierarchical file system that leads to a particular file or directory.

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

prime key
in VSAM, 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
See related-byte address.

related-byte address
the displacement of a record or control interval from the beginning of the file.

Relative-Record Data Set
a VSAM file type whose records are loaded into fixed-length slots and referenced by the record numbers of the slots.

relative-record number
in VSAM in an RRDS, a number that identifies the slot, or data space, and the record contained therein.

RRDS
See Relative-Record Data Set.

RRN
See relative-record number.

sequence field
a field that identifies and provides access to segments in a database. It contains the record's key, which is located in the same position in each record of a key-sequenced data set.

skip sequential access
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
in VSAM, a logical record that is contained in more than one control interval.

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

Virtual Storage Access Method
See VSAM.

VSAM
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 accumulate file usage statistics.

VSAM data set
a classification that indicates how the records in an operating system data set are organized. VSAM is an acronym for Virtual Storage Access Method and is an IBM data access method that provides three ways to organize records in a disk file: Entry-Sequenced Data Set (ESDS), Key- Sequenced Data Set (KSDS), and Relative Record Data Set (RRDS). VSAM allows three types of access to records in VSAM files: sequential, direct, and skip sequential.

VSAM file
See VSAM data set.

Index

G

H

I

K

L

M

N

O

P

R

W