

A Beginner's Guide to Using Random File Access with the SAS System

Glenn Millard
Stephen F. Austin State University

Abstract

One of the SAS System's strengths is its fast sequential file access. However, in many cases, this method does not give the programmer the type of file access he needs. SAS provides various forms of random file access to enable programmers to accomplish their tasks. This type of access has been enhanced greatly in the latest releases of the SAS system. This has left the documentation on this subject scattered over multiple manuals and updates. This paper will present all forms of SAS random file access to provide a beginner with a concise summary on the subject. Various examples of how random file access can be used will be shown as a further aid to someone just beginning to use this SAS feature.

Introduction

Users of the System that are just beginning to write SAS data steps are often unfamiliar with all of the capabilities of the SAS System. SAS procedures and statements can read observations in SAS data sets in the following three patterns: sequential access, random access, and BY-group access. This paper will present the use of random access of SAS data sets. Most SAS procedures and data steps access files sequentially. Random access processes observations according to the value of some indicator variable without processing previous observations. SAS allows random access of observations by observation number, or by an indicator or index.

Index files are familiar to most programmers. Anyone who has worked with databases, ISAM files, standard indexed files in COBOL, or another language are very familiar with indexed files. However, index files are not as commonly used in SAS programming and even those that have used SAS for a number of years may be looking of the manuals when using indexes in the SAS language.

The beginner should understand both the logical and physical makeup of a SAS file or data set. SAS can process data from a variety of sources. To use the processed data, SAS creates a SAS data set, permanent or temporary, that contains the data values and their associated descriptive information. These are the two logical components that all SAS data sets have. They exist in one physical file on the computer. The third logical component a data set can have is one or more indexes. These indexes are kept in a separate physical file.

A SAS index contains the data values of the key variable or variables paired with a location identifier for the observation containing the value. This identifier pair is

contained in a file structure that enables the SAS engine to search by value. Therefore, by using the index the user has created for the file, the SAS System can quickly locate the observations associated with a data value or range of data values. SAS data set variables that are indexed are called key variables. The user can create indexes that are classified as simple or composite. A simple index consists of one key variable. A composite index has multiple SAS variables making up the one key. A single SAS data set can have any number of simple and composite keys associated with it.

All SAS files are organized in libraries. All the files are members of the library and exist together in a library directory. For a SAS file to be processed properly by its index the SAS index and data set must exist in the same SAS library. You can always work with SAS files with host system commands to copy, rename, or delete the files in the library. However, since the index file is separate from the data set, it is possible to make changes this way that will render your index unusable. SAS utilities are available for file management of SAS data sets and other SAS file types. An advantage of learning the utilities is that they work on any host system at any level. The second advantage is that SAS utility procedures automatically copy, rename, or delete any index files associated with a SAS data set.

There are display manager windows available that enable you to perform most file management tasks without submitting SAS program statements. These include the LIBNAME, DIR, VAR, and CATALOG windows. The CATALOG procedure provides all catalog management functions. The CONTENTS procedure lists contents of libraries and data sets, and provide information about their characteristics. The COPY procedure provides statements for copying members of a library. The DATASETS procedure provides management functions for all file types except catalogs. All these utilities help you manager you SAS data sets and their indexes.

When to Use Indexes

Using indexes in SAS require additional computing resources and in some cases will not improve performance. Therefore it is appropriate to discuss when to use indexes in SAS. Indexes provides three major benefits. First, it provides fast access to a subset of the observations in your primary data set. An example of this is in WHERE-expression processing. The second is that BY-group processing can be done without having to sort the data set. The third benefit comes from accessing various observations in secondary data sets that may need to be processed in a random order. The SQL procedure may also use indexes to optimize joining SAS data sets by key variable values.

The user does not control when the SAS System will use the available indexes. The SAS System determines when the index will be used, based on criteria that should determine that using the index will save on computing resources. Generally, the SAS System will use an index when it determines less than one-third of the records will be selected by a WHERE expression. It will not use the index if using it will satisfy all the conditions of the WHERE expression. The SAS System will always use an index for observation retrieval during BY-group processing. It is also recommended that indexes be kept to a minimum for data sets that are updated frequently due to the increased maintenance costs.

Creating Indexes

The first procedure available in SAS for creating indexes was to use the PROC DATASETS utility. The INDEX CREATE statement is used after the MODIFY statement to create an index. The form for creating a simple index is:

```
INDEX CREATE variable;
```

To create a composite index the form is:

```
INDEX CREATE index=(variable-1 ... variable-n);
```

The variables must exist in the data set and the index name for the composite index cannot must be unique; it cannot be the same name as another variable or index in the data set. There are two options available: NOMISS and UNIQUE. The NOMISS option will exclude from the index all observations with missing values for all index variables. The UNIQUE option that the combination of values for the index variables must be unique. If specified, and multiple observations have the same value for the index variables, the index is not created.

An example of a proper DATASETS procedures is shown below. It creates one simple and one complex index on the data set. The CONTENTS statement displays information about the data set and the newly created indexes.

```
proc datasets library=test;
  modify acctlist;
    index create acctnum /nomiss;
    index create name=(firstn lastn) /unique;
  contents data=acctlist;
run;
```

Since the release of 6.07 of the SAS System creating indexes has become much simpler. You don't have to create the data set first and then run Proc DATASETS to index it. You can now create the index at the time the data set is created. This is done by using the INDEX= data set option on the DATA statement of your data step. The NOMISS and UNIQUE options may also be specified while creating the index.

The following statement creates a simple index named ACCOUNT.

```
data acctlist (index=(account));
```

The following statement creates two simple indexes with unique key variables.

```
data acctlist (index=(account/unique city/unique))
```

The following statement creates a composite index called NAME from the variables FIRSTN and LASTN.

```
data acctlist(index=(name=(lastn firstn)));
```

The following creates a simple index called ACCOUNT and a complex index called NAME.

```
data acctlist(index=(account name=(lastn firstn)));
```

The SQL Procedure can also be used to create indexes for SAS data sets.

Further Work with Indexes

Once you have created a data set that has an index you have a number of utilities available to you to help maintain the data set and its indexes. The CONTENTS procedure and the CONTENTS statement of the DATASETS procedure allows you to printout information your data set, including information any indexes the data set may have. An example of a CONTENTS procedure would be:

```
proc contents data=test.acctlist;  
run;
```

An example of the CONTENTS statement of the DATASETS procedure would be:

```
proc datasets library=test;  
  contents data=acctlist;  
run;
```

The COPY procedure is available for copying your data libraries from one location to another or from one file to another. When you copy a data set with the COPY procedure the associated indexes are re-created for the copied data set.

Users need to be aware that you should not use Proc SORT on an indexed data set. It will destroy the indexes if you sort it over itself. If you must sort the data set, sort the file creating a new output data set. Then use the DATASETS procedure to recreate the indexes, and if necessary, then copy or rename the new data set back to the original data set name. You can also delete indexes with the DATASETS procedure using the INDEX DELETE statement which has similar syntax to the INDEX CREATE statement.

WHERE-Expression Processing

Major improvements in WHERE-expression processing of data sets using indexes if the WHERE expression is setup properly. With a simple index like ACCOUNT, that was created in an above example, the following WHERE expressions would be valid:

```
where account in ('12345' '12357' '12386')
where city='Nacogdoches' and account in ('12345' '12357'
'12386')
```

The observations would be retrieved in the indexed order of the variable ACCOUNT. However, in the following WHERE expression the index ACCOUNT will not be used because all the observations for which CITY is NACOGDOCHES may not be included when the index is used.

```
where city='NACOGDOCHES' or account in ('12345' '12357')
```

Therefore, the index ACCOUNT is not appropriate because it will not select all the observations that meet the conditions of the WHERE expression.

The same rules for indexing apply for complex indexes as well. The following WHERE-expression will use the complex indexes, because the conditions are appropriate for selecting all the observations:

```
where lastn between 'Johnson' and 'Smith';
```

However, the following WHERE-expression will not use the index because the variable FIRSTN is not the first variable in the composite index:

```
where firstn in ('John' 'Johnny');
```

BY-Group Processing

In BY-group processing the SAS system will always use an index if an appropriate one exists. Since observations are always retrieved in ascending order; if you specify DESCENDING or NOTSORTED options on the BY statement, the index will not be used. If more than one simple index is specified on the BY statement, the variables specified after the first must be sorted properly within groups or you will get the same error message as you would normally without indexes for processing BY groups. The same result will happen for complex indexes if the observations are not in the proper sorted order for the key variables. If all of the key variables are not specified on the BY statement, or if they are specified in a different order than that of the complex index, then the index will not be used during the process. For further information see Chapter 6 of the SAS Language Reference, Version 6.

Using Indexes

The last way to use indexes in the SAS DATA step is with the SET or MODIFY statements that read observations from SAS data sets. Each time a SET or MODIFY statement is executed an observation is read from the data set. On these statements there are two data set options that allow you to do random or direct access of observations. These are the POINT= and KEY= options. The POINT= option allows you to access an observation by its observation number. The KEY= option allows random access to an observation by a specified simple or composite index.

The syntax for the POINT= option is: POINT=variable-name

The temporary variable specified has the value of the observation you wish to read. This variable is not added to the data set. It cannot be used in conjunction with a BY statement, a WHERE statement, or a WHERE= data set option. Also, because the POINT= option keeps the normal end-of-file indicator from being set, another way of stopping the DATA step must be provided to keep the DATA step out of a possible continuous loop. One or both of the following must be provided when using the POINT= option: 1) a STOP statement, ABORT statement, or another method to stop processing the DATA step, 2) programming logic that checks for an invalid value of the POINT= variable. The automatic variable _ERROR_ is set to 1 when the SAS System attempts to read an invalid value of the POINT= variable. The POINT= option cannot be used with various other forms of SAS data sets; like compressed data sets, transport data sets, data sets on tape, or data set views.

The following is an example of the POINT= option that reads every fourth observation of a data set.

```
data sample;
    do obs=1 to last by 4;
        set mastfile point=obs nobs=last;
        if _error_ then abort;
        output;
    end;
    stop;
run;
```

The syntax for the KEY= option is: KEY=index-name

The index-name is the name of a simple or composite index that has been created for the data set. When using the KEY= statement with either the SET or MODIFY command the automatic variable _IORC_ is created. This automatic variable is set to a return code that shows the status of the most recent I/O operation performed on an observation. Using the SYSRC autocall macro in conjunction with this variable provides you with error-handling information on the most recent I/O operation.

Error-handling is an important part of performing random access of data sets using the KEY= option. The WHERE-expression and BY-group operations access all available matching records. With the KEY= option you are responsible for making sure there is an observation read when a SET or MODIFY statement is executed, or taking appropriate actions when the key does not match an observation.

When a SET or MODIFY statement is executed the automatic variable `_IORC_` receives a value appropriate to the status of the operation. Testing for this value will be discussed below. In addition, if the KEY= value is not found the automatic variable `_ERROR_` is set to 1. If this occurs on a MODIFY statement with a END= option, the temporary variable created for the end-of-file indicator will also be set to 1 when MODIFY cannot find a match for the KEY= value. These indicators should give the programmer all of the information he needs to perform the appropriate error-handling operations.

The automatic variable `_IORC_` contains the best information as to what occurred during the most recent I/O operation. There are currently around 100 possible codes because this automatic variable is used by other features of SAS such as SCL functions, VSAM data step processing, DBMS return codes, and other operations. The best way to test for values of `_IORC_` is with the mnemonic codes provided by SYSRC autocall macro. These values can be tested for with logical IF statements. The following mnemonic codes are the most useful:

`_DSENMR`

The transaction data set observation does not exist on the master data set (used only with MODIFY and BY statements). If consecutive observations with different BY values do not find a match in the master data set, both of them return `_DSENMR`.

`_DSEMTR`

Multiple transaction data set observations with a given BY value do not exist on the master data set (used only with MODIFY and BY statements). If consecutive observations with the same BY values do not find a match in the master data set, the first observation returns `_DSENMR` and the second and following ones return `_DSEMTR`.

`_DSENM`

The data set being modified does not contain the observation requested by the KEY= option.

`_SOK`

The observation was located.

For a more complete list of mnemonics for SAS System return codes look in Chapter 18 of the SAS Screen Control Language, Reference, Version 6, Second Edition. A further discussion on creating and using data sets with indexes is also available in Chapter 5 of this manual.

The behavior of using random file access with SAS data sets when duplicate observations are present in either the transaction, or update, data set or the master, or lookup, data set is important to understand. Without understanding how SAS behaves when duplicate observations are present you may get results that are difficult to understand and debug.

The default behavior of reading an indexed SAS data set is to read the index sequentially until the value of the key changes. If the KEY= value does not change on the next execution of the MODIFY or SET statement, the search begins following the most recently retrieved observation. What does this behavior cause? If you have observations with duplicate keys on the master, or lookup, data set then the second read with the same key value will retrieve the duplicate keyed observation that immediately follows the first in the index. If the duplicate key is in the transaction, or update, data set and the key is unique in the master data set the second read will return a "record not found" error. This is because the index is being searched sequentially and the search for the second read for an observation with the same value is started with the record following the successful first read. If the desired result is to retrieve the unique observation on each consecutive read with the same key value you must specify the UNIQUE option on the MODIFY or SET statement. The UNIQUE option causes the search to always begin at the top of the index for the data file. There are situations where either condition may produce the desired results, therefore it is very important to know whether duplicates can occur or be present in the data and how they should be proceeded by your program. (The UNIQUE option was added in Version 6.07 of the SAS System and is first documented in Technical Report P-242, Changes and Enhancements: 6.08).

Using Indexes: Examples

The following examples will show examples of SET or MODIFY statements with possible error-handling procedures. The examples will work with a master data set called STORES with simple indexes of CITY and STATE and a composite index called LOCATION made up of the variables STATE and ZIP. The transaction data set is called UPDATSTR.

```
data sales.stores;
    set updatstr;
    modify sales.stores key=city;
        if _error_ = 1 then do;
            put 'ERROR occurred, No store in ' city;
            _error_=0;
            stop;
        end;
    sales=sales+newsales;
    exp=exp+newexp;
run;
```

The above example would work just as well with any of the indexes available so long as no duplicate observations exist in the update data set. The error-handling test could also look as follows if the END= option is specified:

```
modify sales.stores key=city end=last;
  if last then do;
    put 'ERROR occurred, No store in ' city;
    last=0;
    stop;
  end;
```

If you wished to test the return code you could use the below code for a bad match:

```
if _IORC_ = %SYSRC(_DSENO) then put 'No store in ' city;
```

and the following for a good match:

```
if _IORC_ = %SYSRC(_SOK) then replace;
```

If you knew the update data set had multiple observations for a location the following code using the UNIQUE option would work:

```
data sales.stores;
  set updatestr;
  modify sales.stores key=city/unique;
  if _error_ = 1 then do;
    put 'ERROR occurred, No store in ' city;
    _error_=0;
    stop;
  end;
  sales=sales+newsales;
  exp=exp+newexp;
run;
```

If the master data set had separate observations for two or more stores, but these stores were in the same city and location with different street addresses; then the records would have duplicate keys. To have SAS use the index and still update the proper observation, the following code could be used.

```

data sales.stores;
    set updatstr;
loop;
    modify sales.stores key=city;
        if _error_ = 1 then do;
            put 'ERROR occurred, No store in ' city '
                on ' nstreet;
            _error_=0;
            stop;
        end;
    if street not = nstreet then go to loop;
    sales=sales+newsales;
    exp=exp+newexp;
run;

```

Conclusion

Random file access with the SAS System can provide many benefits to a SAS programmer. Using random file access in the SAS system is quite easy with the enhancements done in the latest versions of the SAS System. It is hoped that this paper has brought together much of the documentation on this subject. It should also have provided some insight into how the SAS System behaves when accessing observations randomly and that this can be of great help in some circumstances.

Sources

SAS Language Reference, Version 6, First Edition

Chapter 6: pp. 109, 201-204, 214, 217-225, 237-238; Chapter 9: pp. 484-485

SAS Procedures Guide, Version 6, Third Edition

Chapter 12: pp. 193-200; Chapter 14: pp. 203-208; Chapter 17: pp. 247-273

SAS Technical Report P-222, Changes and Enhancements to Base SAS Software, Release 6.07

Summary: pp. xix-xx; Chapter 1: p. 9; Chapter 2: pp. 20-21; Chapter 3: pp. 36-43
Chapter 8: pp. 82-83; Chapter 42: pp. 318-319

SAS Screen Control Language, Reference, Version 6, Second Edition

Chapter 5: pp. 62-63; Chapter 18: pp. 191-199

SAS Technical Report P-242, SAS Software: Changes and Enhancements, Release 6.08

Chapter 1: pp. 1-14, 31-32

ACKNOWLEDGEMENTS

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries.

AUTHOR

The author may be reached at:

Glenn Millard
Stephen F. Austin State University
PO Box 6095
Nacogdoches, Texas 75962
Phone: 409-468-1020
Fax: 409-468-3100
email: gmillard@sfasu.edu