



CSI

INTERNATIONAL

Original Printing03/07/2003
Last Revised03/07/2003

TABLE OF CONTENTS

GETTING STARTED	1
Introduction	3
What OVERRIDE-BUFF Does	4
Installation.....	6
USING OVERRIDE-BUFF	9
OVERRIDE-BUFF Startup Parameters	11
Syntax	11
INIT - Initiation Card.....	12
EXCLUDE - Exclusion Cards	13
DSNAME - Dataset Inclusion Card	15
FILE - File Inclusion Card.....	16
JOB - Job Inclusion Card	17
PROGRAM - Program Inclusion Card.....	19
MESSAGE - Message Specification Card	21
Parameter Interrelationships	22
Generic Names	22
Termination.....	23
Priority Of Buffer Values	23
Close Statistics	24
Guidelines On Selecting Initiation Values.....	24
CICS Considerations.....	27
Miscellaneous Notes	28
APPENDIX A	31
MESSAGES	41
System Console Messages.....	43

TABLE OF CONTENTS

GETTING STARTED

Introduction

VSAM is very important to most installations, yet it is rarely utilized optimally. One consequence of this is that jobs which access VSAM files almost always run longer than necessary.

OVERRIDE-BUFF is a product which is designed to significantly increase the performance of VSAM in every installation. It does this in a way which is transparent to the programs involved, does not alter any VSAM files, and does not make any modifications to VSAM itself. While each installation is different, experience with other installations has shown potential savings to be astounding. The following savings have been achieved in benchmarks of real life applications:

	<u>physical I/Os</u>	<u>elapsed time</u>
typical sequential access	33%	10-50%
typical random access	25-50%	40-60%
clustered random access*	99%	95%

In fact, the performance benefits can be so significant, that it may be possible in some cases to defer the purchase of a new CPU. Perhaps best of all, these savings can be realized almost immediately, with no need to change any existing files, programs or JCL.

Those users who are unfamiliar with VSAM buffering may wish to read the section "A Primer on VSAM Buffering" at this time.

- * This involves accesses to a relatively small area on disk which can be either a small file or a small subset of a large file.

What OVERRIDE-BUFF Does

OVERRIDE-BUFF "front ends" the standard VSAM open module, so that it is invoked just before the file is opened. It examines the ACB (which resides in the program) about to be opened. If VSAM was left to use the default buffer values for the data (BUFND) and index (BUFNI) components, OVERRIDE-BUFF will change these to more appropriate values based on the use of the dataset. If BUFND or BUFNI have been specified either in the JCL or by the program, OVERRIDE-BUFF will do nothing. It is, thus, similar to changing the ACB statements in virtually every VSAM program in an installation. Note that this has no effect upon the file as it resides on disk. When VSAM opens the file, it will see a standard VSAM ACB, and VSAM will process it like any other VSAM file. After the file has been opened, OVERRIDE-BUFF has no further involvement with the file. If the file does not use default values for BUFND and BUFNI, OVERRIDE-BUFF assumes that there is a good reason for using those values and will not modify the ACB.

Note that it is possible that VSAM will use values larger than those specified in OVERRIDE-BUFF parameters (or OVERRIDE-BUFF defaults). This is because VSAM looks at buffer parameters in the catalog entry for a file, and at buffer parameters on the JCL statement for a file, after it looks at the program ACB (whether modified by OVERRIDE-BUFF or not). VSAM takes the largest values it finds as it moves through this sequence. In most cases, OVERRIDE-BUFF values will be the largest but if they are not it may seem as if OVERRIDE-BUFF is not working correctly, particularly using storage to the point of exhaustion (and job abend). In fact, use of large buffer values in the catalog entry for a file or on the DD can cause operational problems (with or without OVERRIDE-BUFF). One reason for using the product is to avoid the problems associated with trying to dedicate large amounts of buffer storage to files via the catalog or DD statement, irrespective of access type or the availability of storage in different jobs, due to program or region size or the number of files used concurrently.

As a result of the improvements brought about by using buffers more efficiently, OVERRIDE-BUFF can be expected to produce the following benefits:

- a reduction in physical I/Os (or EXCPs)
- a reduction in CPU usage
- an increase in throughput

These benefits, unfortunately, come at the expense of:

- an increase in paging
- an increase in region GETMAIN storage requirements

However, it is expected that in almost every installation, the benefits will clearly outweigh the disadvantages.

The best way to determine the effect of OVERRIDE-BUFF is to run several benchmarks, comparing the performance with and without OVERRIDE-BUFF. The following tools should prove helpful:

1. A batch performance monitor product.
2. The CLOSE Statistics provided with OVERRIDE-BUFF. Its use is documented in the "CLOSE Statistics" section.
3. Job accounting reports.

The most important factors to measure are physical I/Os (EXCPs) to VSAM files, CPU time, elapsed time, and paging rate.

In addition to the performance advantages mentioned above, OVERRIDE-BUFF offers the following additional operational benefits:

1. Control over buffering is now placed externally to the programs and, to some degree, external to VSAM. This allows buffer specifications to be easily modified if the environment changes. In this case, the buffer specifications may be easily changed to suit the new environment without having to change a single program or file.
2. VSAM buffer requirements are dynamic. If sufficient region GETMAIN storage is available, it will be used effectively. If not, additional buffers will not be acquired.
3. Some programs (e.g., Easytrieve (Computer Associates)) use an ACB which does not necessarily indicate the type of processing which will actually be performed. No matter what BUFFERSPACE value is used, VSAM's buffer allocation is likely to be less than optimal.

Files accessed using user buffers will show no improvement under OVERRIDE-BUFF. This includes CICS Auxiliary Temporary Storage. These files will be automatically excluded from modification by OVERRIDE-BUFF.

Installation

OVERRIDE-BUFF/MVS is distributed on a tape containing two datasets created by IEBCOPY. The first dataset contains pre-generated load modules to be transferred to your Load Library. The second dataset contains sample JCL and the PRODKEY macro.

Load Library Modules:

FSIBUFST	System startup program.
FSIBUFOP	VSAM open interface program.
FSIBUFCL	VSAM close interface program.
FSIBUFPR	Parameter Parser.

Installation Step 1.

Create the following JCL which will load the distribution tape to your Load Library:

```

//JOBID      JOB (accounting),'LOAD OVERRIDE-BUFF',
//           CLASS=A,MSGCLASS=*,MSGLEVEL=(1,1)
// *
// *  LOAD OVERRIDE-BUFF DISTRIBUTION TAPE
// *
//IEBCOPY EXEC PGM=IEBCOPY
//SYSPRINT  DD SYSOUT=*
//TAPEF1    DD DSN='OVERRIDE.BUFF.LOADLIB',UNIT=TAPE,
//           DISP=OLD,LABEL=(1,SL),VOL=SER=FSISYS
//TAPEF2    DD DSN='OVERRIDE.BUFF.CNTL',UNIT=TAPE,
//           DISP=OLD,LABEL=(1,SL),VOL=SER=FSISYS
//LOADLIB   DD DSN='authorized.linklist.library',DISP=SHR
//SRCELIB   DD DSN='product.source.library',DISP=SHR
//SYSIN     DD *
            COPY I=((TAPEF1,R)),O=LOADLIB
            COPY I=((TAPEF2,R)),O=SRCELIB
// *
//
    
```

Installation Step 2

The programs for OVERRIDE-BUFF must be executed from an **APF Authorized Load Library**. To prevent the need for changes to all of your Batch JCL, the OVERRIDE-BUFF Load Library should also be placed in the Linklist. This means that the library must be listed in both members, **IEAAPFxx** and **LNKLSTxx**, in **SYS1.PARMLIB**.

Installation Step 3

OVERRIDE-BUFF uses an SVC Routing Facility to intercept VSAM OPEN and CLOSE processing. This facility allows one or more BIM/FSI products to intercept OPEN and/or CLOSE processing in a prescribed order. The Routing Facility is implemented transparently in OVERRIDE-BUFF by means of the STRT and STOP parm of program FSIBUFST.

The SVC Routing Facility is shipped separately on a tape labeled: **FSIVSR/MVS**. A cover letter should have been shipped with the tape that will explain how to install it.

Do NOT proceed to Installation Step 4 until you confirm that the SVC Routing Facility has been installed.

Installation Step 4

```
//FSIBUFF JOB          activate OVERRIDE-BUFF
//STEP1  EXEC PGM=FSIBUFST,PARM='STRT'
//FSILIB DD DSN=authorized.linklist.library,DISP=SHR
//FSILPA DD DSN=SYS1.LPALIB(IFG0192A),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN   DD *
(OVERRIDE-BUFF startup parameters)
//
```

The DD statement "FSILIB" must point to the authorized library containing the OVERRIDE-BUFF programs.

The JOB doing the STRT must be authorized for read access to 'SYS1.LPALIB' to verify the DFP and VSAM release levels.

The OVERRIDE-BUFF startup parameters are described in the next section, "Using OVERRIDE-BUFF".

The activation is accomplished by the following procedure:

- The Routing programs of FSIVSR/MVS are loaded into CSA storage and queued onto the CDE chain.

- The unloaded load module IDA0192A is read from SYS1.LPALIB. The CESD and RLD records are scanned for the VCONS for IDA019A and IDA0200T. The addresses of the routers are then inserted into these VCONS.
- The addresses of IDA0192A and IDA0200T are stored in a control section of the router.

Since this process does change internal control blocks within MVS, it is important that you test this in your environment during a non-production time period.

If you experience problems with your system after activating OVERRIDE-BUFF, first attempt to deactivate it using the same JCL used to start OVERRIDE-BUFF, except with a PARM='STOP' on the EXEC for FSIBUFST.

USING OVERRIDE-BUFF

OVERRIDE-BUFF Startup Parameters

When you start OVERRIDE-BUFF, you specify a series of parameter cards that describe which ACBs are to be modified. The parameter cards are described in detail below. They must conform to the following standard syntax rules:

Syntax

- An asterisk in column 1 indicates a comment card.
- The keyword (such as INIT) may be preceded by one or more spaces and must be followed by one or more spaces.
- The parameters must have no imbedded spaces.
- Parameters and values are terminated by a space. Comments may follow.
- Only columns 1-72 are examined.
- A card may be continued by following any parameter with a comma and a space. The continuation may begin at any column on the next card. Sublists may not be continued, that is, the left and right parentheses must be on the same card.

INIT - Initiation Card

The INIT card must be the first parameter specified. It contains the default buffer specifications, and the maximum number of parameter entries that can follow. The format of the INIT card is as follows:

```
INIT BUFND=(sss,rrr),BUFNI=(sss,rrr),
    ENTRIES=nnn
```

The INIT parameters are as follows:

BUFND=(sss,rrr) (default values are 5,0)

This parameter specifies the default number of data buffers to be used. The first (or only) value specifies the number of buffers to be used if sequential access has been specified for the file. The second value specifies the number of buffers to be used if random access has been specified. If both random and sequential access have been specified, the larger of the two values will be used. For those few circumstances where more than one string is used, one additional buffer will be allocated for each additional string. (For COBOL programs, the ACB specifies sequential access when the SELECT clause specifies ACCESS MODE IS SEQUENTIAL or ACCESS MODE IS DYNAMIC.) A zero may be specified to indicate that the default value is to be used. These values may be over ridden for individual files or programs by specifying FILE, DSNAME or PROGRAM cards as described below.

BUFNI=(sss,rrr) (default values are 2,6)

This parameter specifies the default number of index buffers to be used. The first value specifies the number of buffers to be used if sequential access has been specified for the file. The second (or only) value specifies the number of buffers to be used if random access has been specified. If both random and sequential access have been specified, the larger of the two values will be used. For those few circumstances where more than one string is used, one additional buffer will be allocated for each additional string. (For COBOL programs, the ACB specifies direct access when the SELECT clause specifies ACCESS MODE IS RANDOM or ACCESS MODE IS DYNAMIC.) A zero may be specified to indicate that the default value is to be used. These values may be over ridden for individual files or programs by specifying FILE, DSNAME or PROGRAM cards as described below.

ENTRIES=nnn

This parameter specifies the maximum number of parameters that can be specified.

EXCLUDE - Exclusion Cards

EXCLUDE cards may be used for special circumstances where it is desirable to exclude certain files, jobs, and/or programs from processing by OVERRIDE-BUFF. The following formats are supported

```
EXCLUDE FILENAME=(filename1,filename2,...filename)
EXCLUDE DSNAME=(dataset.name1,...dataset.name)
EXCLUDE DSNAMEM=(hiqual1,...hiqualn)
EXCLUDE JOB=(aaaaaaaa,bbbbbbbb,...zzzzzzzz)
EXCLUDE PROGRAM=(aaaaaaaa,bbbbbbbb,...zzzzzzzz)
```

The EXCLUDE parameters are as follows:

FILENAME=(name1,...namen) (default is none)

This card specifies the filenames (the 1-8 character name on the DD card, not the 1-44 character data set name in the VSAM catalog) of files which are never to be modified by OVERRIDE-BUFF. This may be necessary, for example, if a file has a large CI size which would otherwise result in excessive use of region GETMAIN space. A filename may be a generic name as described in the "Generic Names" section below.

DSNAME=(dataset.name1,...dataset.name) (default is none)

This card specifies the 1-44 character cluster names in the VSAM catalog of files which are never to be modified by OVERRIDE-BUFF. This may be necessary, for example, if a file has a large CI size which would otherwise result in excessive use of region GETMAIN space.

DSNAMEM=(hiqual1,...hiqualn) (default is none)

This card specifies the leading characters of the cluster names of groups of datasets to be excluded from OVERRIDE-BUFF processing.

JOB=(aaaaaaaa,...) (default value is no jobs excluded)

This card specifies names of jobs whose files are never to be modified by OVERRIDE-BUFF. This may be necessary, for example, if a program contains a large number of files. A job name may be a generic name as described in the "Generic Names" section below.

PROGRAM=(aaaaaaaa,...) (default is no programs excluded)

This card specifies programs whose files are never to be modified by OVERRIDE-BUFF. This may be necessary, for example, if a program

contains a large number of files. A program name may be a generic name as described in the "Generic Names" section below.

DSNAME - Dataset Inclusion Card

The DSNAME card may be used to customize the operation of OVERRIDE-BUFF for a certain VSAM Cluster.

```
DSNAME dataset.name,ACCESS=xxxxxx,BROWSE=xxxxxx,
      BUFND=(sss,rrr),BUFNI=(sss,rrr)
```

The DSNAME parameters are as follows:

dataset.name **(required)**

This specifies the cluster name of a file which is to receive special processing by OVERRIDE-BUFF.

ACCESS=

ACB
RAN
SEQ

(default value is ACB)

This specifies the method which OVERRIDE-BUFF is to use to determine whether sequential and/or direct processing will be performed. The default value, ACCESS=ACB, is appropriate for the vast majority of programs which accurately indicate their intentions in the ACB. A value of ACCESS=BOTH is useful for those programs which may perform both sequential and direct processing but use an ACB which indicates only one of these two possibilities.

BROWSE= [LONG SHORT] **(default value is LONG)**

Programs which frequently perform short browses will probably suffer an increase in EXCPs if the number of data buffers is increased from the default value of 2 to a value larger than 3. Specifying BROWSE=SHORT for such a program will cause a value of BUFND=3 to be used when a file is opened for both direct and sequential access. (In COBOL programs, this is one which is defined as ACCESS MODE IS DYNAMIC.)

BUFND=(sss,rrr) **(default values are 5,0)**

This specifies the number of data buffers to be allocated for this file. These values override the BUFND value in the INIT card.

BUFNI=(sss,rrr) **(default values are 2,6)**

This specifies the number of index buffers to be allocated for this file. These values override the BUFNI value in the INIT card.

FILE - File Inclusion Card

The FILE card may be used to customize the operation of OVERRIDE-BUFF for a certain file.

```
FILE filename,ACCESS=xxxxxx,BROWSE=xxxxxx,
      BUFND=(sss,rrr),BUFNI=(sss,rrr)
```

Its parameters are as follows:

filename **(required)**

This specifies the ddname of a file which is to receive special processing by OVERRIDE-BUFF. The filename may be a generic name as described in the "Generic Names" section below.

ACCESS=

ACB
RAN
SEQ

(default value is ACB)

This specifies the method which OVERRIDE-BUFF is to use to determine whether sequential and/or direct processing will be performed. The default value, ACCESS=ACB, is appropriate for the vast majority of programs which accurately indicate their intentions in the ACB. A value of ACCESS=BOTH is useful for those programs which may perform both sequential and direct processing but use an ACB which indicates only one of these two possibilities. One such program is Easytrieve (from Computer Associates).

BROWSE= [**LONG** **(default value is LONG)**
SHORT]

Programs which frequently perform short browses will probably suffer an increase in EXCPs if the number of data buffers is increased from the default value of 2 to a value larger than 3. Specifying BROWSE=SHORT for such a program will cause a value of BUFND=3 to be used when a file is opened for both direct and sequential access. (In COBOL programs, this is one which is defined as ACCESS MODE IS DYNAMIC.)

BUFND=(sss,rrr) **(default values are 5,0)**

This specifies the number of data buffers to be allocated for this file. These values override the BUFND value in the INIT card described above.

BUFNI=(sss,rrr) **(default values are 2,6)**

This specifies the number of index buffers to be allocated for this file. These values override the BUFNI value in the INIT card described above.

JOB - Job Inclusion Card

The JOB card may be used to customize the operation of OVERRIDE-BUFF for a certain job.

```
JOB jobname,ACCESS=xxxxxx,BROWSE=xxxxxx,
    BUFND=(sss,rrr),BUFNI=(sss,rrr)
```

Its parameters are as follows:

jobname **(required)**

This specifies the 1-8 character job name. The job name may be a generic name as described in the "Generic Names" section below.

ACCESS=

ACB
RAN
SEQ

(default value is ACB)

This specifies the method which OVERRIDE-BUFF is to use to determine whether sequential and/or direct processing will be performed. The default value, ACCESS=ACB, is appropriate for the vast majority of programs which accurately indicate their intentions in the ACB. A value of ACCESS=BOTH is useful for those programs which may perform both sequential and direct processing but use an ACB which indicates only one of these two possibilities. One such program is Easytrieve (from Computer Associates).

BROWSE= [**LONG** **(default value is LONG)**
SHORT]

Programs which frequently perform short browses will probably suffer an increase in EXCPs if the number of data buffers is increased from the default value of 2 to a value larger than 3. Specifying BROWSE=SHORT for such a program will cause a value of BUFND=3 to be used when a file is opened for both direct and sequential access. (In COBOL programs, this is one which is defined as ACCESS MODE IS DYNAMIC.)

BUFND=

(sss,rrr)
NO
YES

(default is BUFND=YES)

This specifies the number of data buffers to be used for all files used by the job. These values override the BUFND value in the INIT card described above. A numeric value takes precedence over the corresponding value for any of the job's files which are (also) specified by FILE or DSNNAME cards. A value of BUFND=NO may be specified to indicate that only index buffers are to be overridden; this reduces the GETMAIN requirements while retaining the buffering advantages for the index component. If omitted or specified as BUFND=YES, the values from the INIT, FILE or DSNNAME card will be used.

BUFNI = $\left[\begin{array}{l} (sss, rrr) \\ \text{NO} \\ \text{YES} \end{array} \right]$ **(default is BUFNI=YES)**

This specifies the number of index buffers to be used for all files used by the job. These values override the BUFNI value in the INIT card described above. A numeric value takes precedence over the corresponding value for any of the job's files which are (also) specified by FILE or DSNNAME cards. A value of BUFNI=NO may be specified to indicate that only data buffers are to be overridden; this reduces the GETMAIN requirements while retaining the buffering advantages for the data component. If omitted or specified as BUFNI=YES, the values from the INIT, FILE or DSNNAME card will be used.

PROGRAM - Program Inclusion Card

The PROGRAM card may be used to customize the operation of OVERRIDE-BUFF for a certain program.

```
PROGRAM progname,ACCESS=xxxxxx,BROWSE=xxxxxx,
      BUFND=(sss,rrr),BUFNI=(sss,rrr)
```

Its parameters are as follows:

progname **(required)**

This specifies the 1-8 character phase name. The program name may be a generic name as described in the "Generic Names" section below.

ACCESS= ACB
RAN
SEQ **(default value is ACB)**

This specifies the method which OVERRIDE-BUFF is to use to determine whether sequential and/or direct processing will be performed. The default value, ACCESS=ACB, is appropriate for the vast majority of programs which accurately indicate their intentions in the ACB. A value of ACCESS=BOTH is useful for those programs which may perform both sequential and direct processing but use an ACB which indicates only one of these two possibilities. One such program is Easytrieve (from Computer Associates).

BROWSE= [LONG
SHORT] **(default value is LONG)**

Programs which frequently perform short browses will probably suffer an increase in EXCPs if the number of data buffers is increased from the default value of 2 to a value larger than 3. Specifying BROWSE=SHORT for such a program will cause a value of BUFND=3 to be used when a file is opened for both direct and sequential access. (In COBOL programs, this is one which is defined as ACCESS MODE IS DYNAMIC.)

BUFND= (sss,rrr)
NO
YES **(default is BUFND=YES)**

This specifies the number of data buffers to be used for all files used by the program. These values override the BUFND value in the INIT card described above. A numeric value takes precedence over the corresponding value for any of the program's files which are (also) specified by FILE or DSNAME cards. A value of BUFND=NO may be specified to indicate that only index buffers are to be overridden; this reduces the GETMAIN requirements while retaining the buffering advantages for the index component. If omitted or

specified as BUFND=YES, the values from the INIT, FILE or DSNAME card will be used.

BUFNI = $\left[\begin{array}{l} (sss, rrr) \\ \text{NO} \\ \text{YES} \end{array} \right]$ (default is BUFNI=YES)

This specifies the number of index buffers to be used for all files used by the program. These values override the BUFNI value in the INIT card described above. A numeric value takes precedence over the corresponding value for any of the program's files which are (also) specified by FILE or DSNAME cards. A value of BUFNI=NO may be specified to indicate that only data buffers are to be overridden; this reduces the GETMAIN requirements while retaining the buffering advantages for the data component. If omitted or specified as BUFNI=YES, the values from the INIT, FILE or DSNAME card will be used.

MESSAGE - Message Specification Card

The MESSAGES card may be used to produce messages for certain events. Normally, OVERRIDE-BUFF will produce no messages as it processes an ACB. However, specifying a value of "YES" for the following parameters will cause an informative message to be issued each time the associated event occurs. Most installations will want to run with all messages off once OVERRIDE-BUFF is running in production; however, turning some or all of the messages on may help in monitoring the operation of OVERRIDE-BUFF during the initial testing stages.

```
MESSAGES BUFND=xxx , BUFNI=xxx , FILEEXCL=xxx  
MESSAGES JOBEXCL=xxxx , MODIFIED=xxxx , PROGEXCL=xxxx , USERBUFF=xxxx
```

The parameters are:

BUFND=YES	ACB not modified because BUFND was explicitly set in the ACB
BUFNI=YES	ACB not modified because BUFNI was explicitly set in the ACB
FILEEXCL=YES	ACB not modified because the file was excluded or a buffer override value of zero was specified
JOBEXCL=YES	ACB not modified because the job was excluded
MODIFIED=YES	ACB was modified
PROGEXCL=YES	ACB not modified because the program was excluded
USERBUFF=YES	ACB not modified because it specifies that the program will supply its own buffers

Parameter Interrelationships

OVERRIDE-BUFF allows BUFND and BUFNI specifications on several of its parameter cards. They are processed in the following order, from the highest to the lowest priority:

- If the ACB indicates that the program supplies its own buffers, the ACB will not be modified.
- If an EXCLUDE card applies to the file, program, or job, the ACB will not be modified.
- If BUFND=NO is specified on a matching JOB or PROGRAM card, BUFND will not be modified.
- If BUFND=nnn and/or BUFNI=nnn is specified on a FILE or DSNNAME card, the specified values, will be used, when appropriate, to override the values in the ACB.
- The BUFND=nnn and/or BUFNI=nnn values (or their defaults) on the INIT card will be used.

Generic Names

The file names, job names and program names on the various initialization cards may be generic names if desired. To specify a generic name, use a "+" for any of the characters to indicate that any value for that character is to be accepted. Any unspecified characters to the right of the name are assumed to be significant spaces, not "+" characters. For example, a program name of AP++++ will match AP1000, APPROG and AP400, but not AP2100TS.

By placing the initialization cards in a specific sequence, it is possible to make certain exceptions to the generic names. For example, assume that the following requirements must be met:

- Most GL files are to use BUFND=3,BUFNI=5
- The GLA files are to use BUFND=5,BUFNI=10
- File GLA100 is to use BUFND=0,BUFNI=20
- FILE GLAHOG is to be excluded entirely
- All test files (including GL) are to use BUFND=0,BUFNI=3

All of the above conditions may be met by specifying the initialization cards in the following sequence:

```
FILE +++TEST, BUFND=0, BUFNI=3
FILE GLAHOG, BUFND=0, BUFNI=0 (or EXCLUDE FILENAME=GLAHOG)
FILE GLA100, BUFND=0, BUFNI=20
FILE GLA++++, BUFND=5, BUFNI=10
FILE GL+++++, BUFND=3, BUFNI=5
```

Termination

OVERRIDE-BUFF may be terminated at any time and from any region using a job similar to the following:

```
//FSIBUFF JOB          terminate OVERRIDE-BUFF
//STEP1 EXEC PGM=FSIBUFST, PARM=' STOP '
//SYSPRINT DD SYSOUT=*
//
```

Priority Of Buffer Values

Due to the diverse sources of buffer values, this section describes the priority by which a file's buffer values (BUFND and BUFNI) are selected.

- The ACB will not be modified by OVERRIDE-BUFF if the relevant file or job was EXCLUDED during initialization.
- If a non-zero value was specified in the ACB by the program, that value will not be modified by OVERRIDE-BUFF.
- If a matching PROGRAM card was specified and it uses a non-zero buffer value, that value will be used. If a buffer value of zero was explicitly specified, the ACB will not be modified regardless of the values specified on the FILE, DSNAMES and INIT cards.
- If a matching FILE or DSNAMES card was specified and it uses a non-zero buffer value, that value will be used. If a buffer value of zero was explicitly specified, the corresponding value from the INIT card will be used.
- The value from the INIT card will be used if it is non-zero.
- VSAM will use the MAXIMUM value derived from the specifications in the ACB (whether modified by OVERRIDE-BUFF or not), the catalog, and the DD statement.

Close Statistics

OVERRIDE-BUFF can generate a series of statistics messages to a FSISTAT DD each time a VSAM dataset is closed. These messages can be used to show the effect that OVERRIDE-BUFF has upon physical I/Os. It does not in any way affect the operation of OVERRIDE-BUFF or VSAM and may be used either with or without OVERRIDE-BUFF.

The messages are generated for any job that contains a FSISTAT DD in its JCL, while OVERRIDE-BUFF is active in your system. The following messages are generated:

```

FSIBUFF VER 3.0x CUMMULATIVE STATISTICS FOR DATA SET... WBYEAM.FSIJMSTR.CNTLFILE
FSIBUFF TOTAL AMOUNT OF STORAGE USED FOR DATA BUFFERS.                20480
FSIBUFF TOTAL AMOUNT OF STORAGE USED FOR INDEX BUFFERS.                 512
FSIBUFF NUMBER OF INDEX LEVELS.....                                  1
FSIBUFF NUMBER OF EXTENTS IN THE DATA SET.....                       1
FSIBUFF NUMBER OF USER-SUPPLIED RECORDS IN THE DATA SET              7
FSIBUFF NUMBER OF DELETED RECORDS.....                               0
FSIBUFF NUMBER OF INSERTED RECORDS.....                              0
FSIBUFF NUMBER OF UPDATED RECORDS.....                               0
FSIBUFF NUMBER OF RETRIEVED RECORDS.....                             1358
FSIBUFF NUMBER OF BYTES OF FREE SPACE IN THE DATA SET..              19968
FSIBUFF NUMBER OF TIMES A CONTROL INTERVAL WAS SPLIT...               0
FSIBUFF NUMBER OF TIMES A CONTROL AREA WAS SPLIT.....                 0
FSIBUFF NUMBER OF TIMES A I/O REQUEST WAS ISSUED.....                 2711
FSIBUFF PERCENTAGE OF FREE CI IN THE CA .....                          0
FSIBUFF PERCENTAGE OF FREE BYTES IN THE CI .....                      0
FSIBUFF NUMBER OF CI IN A CA .....                                    46
FSIBUFF NUMBER OF FREE CI IN A CA .....                               0
FSIBUFF NUMBER OF FREE BYTES IN A CI .....                            0
FSIBUFF CI SIZE .....                                                512
FSIBUFF MAXIMUM RECORD SIZE .....                                    505
FSIBUFF EXCP COUNT FOR THIS OPEN .....                                22
    
```

Most of the information printed is the same information that would be displayed in an IDCAMS LISTCAT. Except where stated, the information printed is for the Data Component of the VSAM Cluster.

Guidelines On Selecting Initiation Values

OVERRIDE-BUFF is designed to produce a large, immediate performance improvement. This should not, however, eliminate the need for tuning the VSAM files themselves. The CISZ, SPEED, IMBED, FREESPACE and VOLUMES parameters, for example, can have significant performance ramifications and should be specified carefully.

It is important to keep in mind that OVERRIDE-BUFF treats all files alike unless you specify otherwise.

The first objective when using OVERRIDE-BUFF should be to be run for at least 24 hours without running out of region GETMAIN storage.

This is covered in detail in the section "How to Deal With Insufficient GETMAIN Storage." (By the way, GETMAIN problems are not inevitable, but it is a good idea to be prepared in case it happens.) During this stage, files and programs with large GETMAIN requirements under OVERRIDE-BUFF should be identified and dealt with. Keep in mind the following guidelines:

- The best situation, where possible, is to provide plenty of region GETMAIN storage.
- The EXCLUDE card should be used either as a quick fix or as a last resort. Generally, it is better to use other techniques on a permanent basis, such as:
 - a. reducing the file's CI size if it is too large
 - b. lowering the BUFNI and/or BUFND values on the INIT, FILE, or PROGRAM cards
 - c. specifying BUFND=NO on the JOB or PROGRAM cards
- The number of files, jobs and programs to be treated specially should be kept as small as is reasonably possible.

The second objective is to optimize the operation of OVERRIDE-BUFF. This will require some amount of experimentation. Some installations may not be willing to invest the time to further refine the operation of OVERRIDE-BUFF. However, additional performance gains can be achieved by doing so. At this point, a batch performance monitor can prove valuable in measuring the effects of any changes. There are three basic areas to be addressed:

- The general parameters specified on the INIT card.
- The specific parameters on the EXCLUDE, FILE, JOB, and PROGRAM cards.
- Factors which are not directly related to OVERRIDE-BUFF such as placement on disk volumes, FREESPACE, and Control Area size.

When specifying values for the general parameters on the INIT card, keep in mind the following guidelines:

- Generally, these values should be specified as generously as practical in an attempt to maximize the advantages of VSAM buffering. A small, but reasonable number of exceptions may be specified by using the EXCLUDE, FILE, JOB and PROGRAM cards where it is necessary to be less generous (or perhaps even more generous).

- The BUFNI parameter applies to randomly (direct) accessed Key Sequenced (KSDS) files only. The BUFNI parameter tends to be more effective at reducing I/Os than the BUFND parameter. In most cases, the index CI size should be either 512 or 1024 bytes. This makes index buffers not only good for performance but very inexpensive as far as storage requirements are concerned. If a file has been DEFINED with a larger index CI size, it would probably be a good idea to reduce it. The best way to do this is to DEFINE the file with the CISZ parameter specified only at the DATA level, not at the CLUSTER or INDEX level, thus allowing VSAM to choose an appropriate value for the index CI size. To cause VSAM to select a small index CI size, specify large CI and (indirectly) large CA sizes for the data component.
- The BUFND parameter applies to all types of VSAM files. Although its primary usage is for sequential access, it can also affect random access. The biggest concern here is with the size of the data CI. It is very easy to require a large amount of GETMAIN storage if the data CI size is large. Be sure that a large data CI size is used only if it is appropriate. There is an unexpected distinction between using 3 data buffers and using 4 data buffers when a file is accessed sequentially. If 4 or more data buffers are specified, VSAM will divide them into two halves and use a "double buffering" technique when performing sequential I/O; if 3 or fewer data buffers are specified VSAM will perform I/O using all buffers at one time. The "double buffering" technique offers the advantage of making records available with little or no delay, whereas the "single buffering" technique offers the advantage of minimizing the number of I/Os performed. In today's typical installation with many regions active simultaneously, it should be possible to minimize GETMAIN requirements while still reducing I/Os by using exactly 3 data buffers; in fact, some installations may find that overall system performance is enhanced better by using 3 than by using 4 or 5 data buffers.
- When trying to reduce the virtual storage requirements, it is probably best to attempt to keep BUFNI as high as possible and achieve any savings by lowering the BUFND value. In extreme cases, the BUFND value can be reduced to zero, but this precludes any performance gain for the data component. This can be accomplished by adjusting:
 - a. the BUFND and BUFNI parameters on the INIT, FILE, and PROGRAM cards
 - b. the BUFND=NO parameter on the JOB and PROGRAM cards
- After having established appropriate BUFND and BUFNI values, some adventurous installations may wish to experiment with unusually large values for some or all files. If sufficient storage is available, additional performance gains may be observed when using 10, 50 or more buffers.

When specifying values for the specific parameters on the EXCLUDE, FILE, DSNNAME, JOB, and PROGRAM cards, keep in mind the following guidelines:

- When possible, it is better to use OVERRIDE-BUFF a little bit than to eliminate it. Thus, the FILE, DSNNAME, JOB and PROGRAM cards are to be preferred to the EXCLUDE card.
- The FILE, DSNNAME, JOB and PROGRAM cards can be useful, but they require some work. The best (and maximum recommended) value for BUFNI for a random (direct) KSDS file is:

index set size + 1 + STRNO (STRNO is usually 1)

The index set size is not shown directly in a LISTCAT report but can be determined by subtracting the number of data CAs from the number of index CIs. If the BUFNI value calculated above is too large, a good alternative is:

number of index levels + 1 + STRNO (STRNO is usually 1)

The number of index levels is shown by the IDCAMS LISTCAT output. It can be surprising how few records an index actually occupies for a moderate size file.

- Concentrate your efforts on optimizing the files (and programs) which are most heavily used.

CICS Considerations

OVERRIDE-BUFF treats CICS like any other region: it modifies those ACBs which use the default values and ignores those ACBs which use other values. Thus, OVERRIDE-BUFF may be used for CICS if desired. However, CICS file processing requirements are often very different from batch requirements. Therefore, it is usually advantageous to carefully tune each individual VSAM file and specify specific values for the BUFND, BUFNI and STRNO parameters in the DFHFCT TYPE=ENTRY macros. In this case, OVERRIDE-BUFF will never see a file which uses the default values and will never modify a CICS VSAM file's ACB. If an installation has files which use the default buffer specifications, OVERRIDE-BUFF may certainly be used to quickly increase those values. Users are strongly encouraged to explicitly define all buffer specifications in the FCT rather than simply allowing OVERRIDE-BUFF to increase them.

If a CICS file uses VSAM Shared Resources (specified by DFHFCT TYPE=DATASET,SERVREQ=(SHARE,...) in the FCT), VSAM will ignore the buffer specifications in the ACB. Thus, the presence or absence of OVERRIDE-BUFF will have no effect upon the operation of the file.

CICS may use a number of files which are not defined in the FCT. OVERRIDE-BUFF may be used for some of these files, but for others for which CICS performs its own buffering, such as DFHNTRA and DFHTEMP, OVERRIDE-BUFF will have no effect.

It is worth noting that OVERRIDE-BUFF will indirectly improve the performance of a CICS region by reducing the demands for I/O and CPU placed upon the system by batch jobs. This will be evidenced by decreased service times for CICS I/O and faster response times. This improvement in response times may (or may not) produce a measurable increase in the amount of productive work that is performed by CICS; if this is the case, a corresponding increase in CICS CPU usage, transaction counts, file I/Os, etc. should be construed to be a benefit, not a detriment.

Miscellaneous Notes

- There are a number of times when IBM products will explicitly specify the number of buffers to be used:
 - a. IDCAMS: BACKUP/RESTORE, EXPORT/IMPORT
 - b. COBOL uses BUFND=5 when initially loading a file

In these cases, OVERRIDE-BUFF assumes that the program uses the value for a specific reason and does not modify the ACB.

- When processing sequentially for a file defined with SHAREOPTIONS(4), VSAM does not perform read-ahead I/O. Such files should be defined with a FILE card such as:

```
FILE xxxxxxx,BUFND=n
```
- Where "n" is the number of index levels, which is almost always either 2 or 3. This will cause OVERRIDE-BUFF to use the default number of data buffers when the file is accessed sequentially (thus avoiding wasting GETMAIN storage on unused buffers) but increase the number of index buffers when the file is accessed randomly.
- For a detailed discussion of VSAM, see the IBM manual VSAM Primer and Reference G320-5774.

APPENDIX A

A Primer On VSAM Buffering

This section is included as an introduction to VSAM buffering for those who are unfamiliar with the way VSAM handles buffers. It does not cover all of the intricacies of VSAM buffering. Rather, it serves to illustrate some of the strengths and weaknesses in VSAM buffering as distributed by IBM and how OVERRIDE-BUFF can help VSAM to run better.

VSAM Buffer Description

VSAM was introduced in the early 1970's along with the new Virtual Storage operating system, OS/VS. One of the reasons for its name, Virtual Storage Access Method, was that it was designed to take advantage of this new facility called virtual storage. It did this by allowing data to be kept in buffers located in virtual storage rather than performing a disk I/O to retrieve each block of data as needed. This offered a significant performance advantage when a record must be read or written more than once, since there is a chance that the record is still located in a buffer and only the first physical I/O may be necessary; the subsequent I/O(s) may be bypassed. This was especially advantageous for the index component of a Key Sequenced Data Set (KSDS), since all random reads and writes must read the index component as well as the data component.

These buffers for VSAM files were placed, not in the program itself, as the older access methods had done, but were placed in separate GETMAINed areas. These areas are also used to hold other control blocks used by VSAM. Each buffer contains exactly one Control Interval. When using a KSDS, there are separate buffers for the data component and for the index component.

VSAM Buffer Use Example

To illustrate how VSAM makes use of buffers, let's take a simple example of a program which is randomly reading a KSDS file. This file has two index levels (which is typical of small to medium size files). The first index level consists of a single Control Interval (CI). This CI points, not to data records, but to index records in the second index level. These second level index records point to the actual data records. Let us assume that the program has requested that this file be processed using three index buffers and two data buffers. The first read will require the following physical I/O operations:

- The first level index CI is read. This tells which second level index CI is required.
- The required second level index CI is read. This tells which data CI is required.
- The desired data CI is read.

Thus, the first read issued by the program has resulted in three physical I/Os. What happens with the next read? Because the program specified that several buffers were to be allocated, it is possible for the next read request to be satisfied with no physical I/Os if it is for a record which is located in the same data CI. This would make it possible to satisfy the second read almost immediately since no I/O would be required.

But what happens if a read is requested for a different part of the file? The result depends upon which buffers are left in storage from previous read(s). In this case, VSAM proceeds as follows:

- The first index level CI is still in virtual storage, so no I/O needs to be performed in order to access it. This will always be true as long as more than one index buffer is allocated.
- If the second level index CI required for this read is still in storage no I/O needs to be performed to retrieve it. If a different CI is required, it will be necessary to perform another physical I/O. Since we have specified that there are to be three index buffers, and only two of them have been used so far, the third buffer is available for use to contain this index CI. If all of the buffers are in use, VSAM will be forced to reuse one of them.
- The required data CI is read into storage.

Thus, for this second I/O, anywhere from zero to two physical I/Os will be performed. With these buffer specifications, only the first read request will require the maximum number of I/Os; all subsequent read requests will require fewer. VSAM will continue this process for all of the read requests, keeping as much data in the buffers as possible.

The actual process used by VSAM is not as simple, nor as efficient, as one might hope, but the above example illustrates the basic process. From this example, we can draw the following conclusions:

- The more buffers we have allocated, the more CIs VSAM is able to process without performing physical I/Os.
- When performing random I/O to a KSDS, it is possible to have more physical I/Os to the index component than to the data component. Thus, index buffers may be more important than data buffers.

When performing sequential reads, VSAM operates slightly differently. When the first read is issued, VSAM will perform one physical I/O to fill either all or half of the data buffers, thus "reading ahead" as far as possible. (If more than 3 data buffers are allocated, VSAM uses a "double buffering" technique, filling half of the buffers with each read, otherwise it fills all of the buffers in a single read.) As the program calls for logical records one at a time, VSAM will access the next logical record from a CI until the end of the CI is reached. If the next CI is located in a buffer, VSAM will immediately access the logical record in the next CI without waiting for a physical I/O. At this point, VSAM does not perform another "read ahead" to fill the buffer which has just been emptied. When all of records in either all or half of the buffers have been processed, VSAM will again read CIs from disk into all remaining data buffers at one time. This, again, allows VSAM to minimize the number of physical I/Os which are performed.

Before further examining the effect that buffering has upon performance, let's determine approximately how much storage is required for all of these buffers. Since a buffer holds one CI, the amount of buffer storage needed depends upon the size of the CI. An additional 100 bytes (approximately) will also be needed for other control blocks to manage each buffer. If the file in our example has a data CI size of 4096 bytes and an index CI size of 1024 bytes (which are reasonably typical values and probably reasonably efficient), the buffers will occupy:

index: $3 \text{ buffers} * 1024 \text{ bytes} = 3072 + 300 = 3372 \text{ bytes}$

data: $2 \text{ buffers} * 4096 \text{ bytes} = 8192 + 200 = 8392 \text{ bytes}$

Thus, this file will use approximately 11,800 bytes (or slightly less than 12K) of virtual storage for buffers. If a program processes four files with similar characteristics, 48K of storage will be required for buffers. (Remember that VSAM requires storage for other areas as well.) When compared to a typical region size, this amount seems rather modest.

Estimating Performance Considerations

As we have seen above, using many buffers can produce a reduction in physical I/Os. I/Os are relatively expensive and are often a major bottleneck to system performance. How many I/Os can we expect to save? While each program and file are different, let's return to our sample random access program above. Instead of using three index buffers and two data buffers, the program could have specified the minimum number of buffers, which is:

BUFNI: 1

BUFND: 2

If the file is accessed using the minimum of one index buffer and one data buffer, each read will require 2-3 physical I/Os, two for index records and one or none for the data records. 1000 reads would require 2000-3000 I/Os. But if we allow enough buffers, the first read would require 3 I/Os, and the subsequent reads would require no more than two and often no I/Os. In this case 1000 reads might require 500 I/Os. In an extreme case, but which does happen in the real world, where the program reads the records in clusters which occupy only a few number of CIs, all of which are in buffers, it would be possible to perform 1000 reads with perhaps as few as 10 I/Os. This type of savings can have a drastic positive effect upon system performance. It is not uncommon to see the run times of certain programs reduced to a fraction of their old times merely by implementing good VSAM buffer values.

A little known fact about physical I/Os is that processing one I/O requires thousands of CPU instructions by the access method (VSAM) and the Supervisor. While the CPU is relatively very fast compared to the speed of an I/O operation, the cumulative effect of all of these CPU instructions can produce a measurable effect on run times. Furthermore, while many I/Os can be in progress at one time (as many as one per device), there is only one CPU, and wasteful use of the CPU will hurt all users.

Putting all of these considerations together, let us return to our sample and make some rough assumptions from which we can make some rough preliminary performance estimates:

- The program is a rather simple batch job which reads a file randomly and produces a report.
- The program divides its time evenly between reading the file and printing the report. It prints one line for each record read. We will assume the time required to process one record and print one print line is 10 milliseconds, or 100 records per second.
- VSAM takes 15 seconds to open the file, close the file, and perform all processing except for the physical I/Os for our 3000 reads.
- One disk I/O takes 50 milliseconds, or 20 I/Os per second.
- Our CPU is capable of executing 1 million instructions per second (MIPS), and each disk I/O requires 5000 CPU instructions. Therefore, the CPU time needed to process an I/O is 5 milliseconds.

Let us compare three different buffering possibilities and examine the effects upon I/Os, CPU time and elapsed time.

	Using the minimum buffers	Using efficient buffers with scattered reads	clustered reads
Reads issued	3000	3000	3000
I/Os performed	2500	500	10
I/O time	125 sec	25 sec	.5 sec
VSAM CPU time for I/Os	12.5 sec	2.5 sec	05 sec
VSAM CPU other time	15 sec	15 sec	15 sec
Program time	30 sec	30 sec	30 sec
Total elapsed time	183 sec	73 sec	46 sec

From this rough estimate, it should be obvious that efficient buffering can significantly improve every factor related to VSAM I/O. The only factors unaffected by VSAM buffering are those unrelated to VSAM, such as the time the program spends in preparing a print line. Furthermore, these savings make more of the CPU (and channel capacity) available to other jobs so even non-VSAM jobs can run faster.

Trade-offs

But, perhaps without knowing it, we have made a trade-off. While the use of many VSAM buffers has reduced physical I/Os and CPU time, it has increased the virtual storage requirements. And, since more virtual storage has been used, we can expect that the paging rate will be increased. The question to be determined is, "Is this a good trade-off?"

For our sample file above, if the minimum buffer specifications had been used (one index buffer and two data buffers), the buffers would require 9K of storage (1 * 1K for the index plus 2 * 4K for the data). Increasing this to five index buffers and two data buffers, which could significantly reduce I/Os, would only increase the buffer requirements to 13K (5 * 1K for the index plus 2 * 4K for the data). If the storage occupied by the program itself plus VSAM and any other access methods is about 100K, this represents an increase in the total virtual storage requirements of about 4%. Surely, the addition of 4K would be unlikely to increase the paging rate significantly. However, if this is done for several files across several regions, the increase in total virtual storage is no longer negligible; however, it is still very likely to be a worthwhile price for the expected savings in I/O and CPU usage.

By increasing a file's buffer specifications we can expect to shift many VSAM I/Os to a few additional paging I/Os. An important advantage of this technique is that paging I/Os are the most efficient I/Os in the system. While the system paging rate can be expected to increase, hopefully by a small amount, we can look at this in two ways. First, it is a cost associated with using a better buffering technique. Second, it is a shifting of resources from expensive VSAM I/Os to cheap paging I/Os. Unless the system has a very high paging rate to begin with, it appears that we have made a very good trade-off.

But there does come a point of diminishing returns. Imagine a file with 250 index buffers and 250 data buffers. Even if there is enough virtual storage available to contain all of these buffers, it is highly unlikely that an active system will be able to keep them all in real storage. As VSAM attempts to examine each of them, the paging rate might go through the roof. Except for very special circumstances, this is an obvious case of excess.

The other trade-off to be considered is the additional virtual storage required. This storage must come from the region's GETMAIN area. While it sounds like a simple idea to merely add more GETMAIN storage, this is often difficult to do and must be handled carefully. In this, each installation is different, and each solution will be a little different.

Strings

VSAM provides the ability for a file to be simultaneously accessed at more than one place. For each string defined for a file, the program may simultaneously access the file in one location. Multiple strings are typically used for online systems (such as CICS) and are used only rarely for batch programs. When a file has multiple strings, the buffers are more or less shared among the strings. The above example has assumed the use of just one string.

The Surprising Conclusion

We have now examined the rationale behind VSAM's use of buffers, and we have shown that there can be significant performance improvements by using several buffers. However, as distributed by IBM, there are two factors which almost totally defeat these potential benefits:

- The default values used by VSAM are designed to minimize the use of virtual storage. Thus, the default buffer values are also the minimum values, and all of the potential benefits will only be achieved if the default values are consciously overridden.
- COBOL and RPG programs cannot specify buffer values and will be guaranteed to receive the default (i.e. minimum) buffer values. While it is possible to specify buffering in the AMP operand of the DD statement or BUFFERSPACE when DEFINEing a file, this method has a number of disadvantages:
 - a. They become fixed requirements; if the specified storage is unavailable, the job will be canceled.
 - b. Modifying buffer requirements may require many JCL changes.
 - c. If the CI size is ever changed, the buffer space parameter(s) should be changed manually.

OVERRIDE-BUFF is designed to remedy all of the above problems and add some additional benefits

MESSAGES

System Console Messages

The following messages may appear on the job log:

FSIBUFF-100 **[ACBBUFND
ACBBUFNI]** FOR dataset.name MODIFIED TO n
(See below).

FSIBUFF-100 EXCLUDE FOR dataset.name PER **[PGM
DSN
JOBNAME
FILENAME]** xxxxxx

One or more of these messages is displayed depending on the MESSAGE card operands that have been specified.

FSIBUFF-101 **FSIBUFTB TABLE NOT FOUND**
OVERRIDE-BUFF is not currently running.

FSIBUFF-102 **NO FSISTAT DD CARD FOUND**
Statistics will not be printed, as this DD statement is not in the JCL.

FSIBUFF-103 **GETMAIN ERROR, (error description)**
OVERRIDE-BUFF cannot be initiated due to a failure of its GETMAIN macro.

FSIBUFF-104 **SYSIN ERROR, (error description)**
An invalid or misspelled parameter has been specified.

FSIBUFF-105 **NO PARMS SPECIFIED ON EXECUTE CARD**
You must specify one of the following parms on the EXEC statement: STRT, STOP, or REFR.

FSIBUFF-106 **INVALID PARM SPECIFIED**
An invalid or misspelled keyword has been specified. The keyword is the first character string on the card, such as "INIT"

FSIBUFF-107 **FREEMAIN FAILED, (error description)**
A FREEMAIN macro issued by OVERRIDE-BUFF has failed. This should never occur.

FSIBUFF-108 **TABLE NOT REFRESH, NOT LOADED**
OVERRIDE-BUFF tried to refresh the FSIBUFTB table, but it was not loaded. Use the PARM=STRT option to activate OVERRIDE-BUFF.

FSIBUFF-109	TABLE ALREADY LOADED An attempt was made to initiate OVERRIDE-BUFF, but it is currently running. Use the REFR PARM if you want to load a new set of OVERRIDE-BUFF parameters.
FSIBUFF-110	CDE LOAD ERROR Request technical assistance for this error.
FSIBUFF-111	CDE UNLOAD ERROR Request technical assistance for this error.
FSIBUFF-112	ERROR RETURNED FROM ROUTER xxxx Request technical assistance for this error.
FSIBUFF-113	SUBPOOL NOT FOUND Request technical assistance for this error.
FSIBUFF-114	MODULE xxxxxxxx NOT FOUND Required module 'xxxxxxx' could not be loaded. Refer to the installation section for proper load library contents and setup.
FSIBUFF-115	ERROR RETURN FROM FSIBUFPR Request technical assistance for this error.
FSIBUFF-116	OVERRIDE-BUFF RELEASE 3.0A SUCCESSFULLY xxxxxxxx The specified version of OVERRIDE-BUFF has been successfully initiated, stopped or refreshed.
FSIBUFF-117	NO TCT ENTRY FOUND Request technical assistance for this error.
FSIBUFF-9998	WARNING - PRODUCT AUTHORIZATION EXPIRES IN 30 DAYS OR LESS Every version of OVERRIDE-BUFF is distributed with a built in expiration date. This message will be issued if OVERRIDE-BUFF is initiated within 30 days prior to the expiration date at which time OVERRIDE-BUFF will refuse to run.
FSIBUFF-9999	PRODUCT AUTHORIZATION HAS EXPIRED The expiration date of OVERRIDE-BUFF has been reached. It will be necessary to order a current version of OVERRIDE-BUFF or to obtain a temporary patch in order to continue to use OVERRIDE-BUFF.