

O v e r r i d e - B U F F / V S E

DYNAMIC VSAM BUFFERING

PROGRAM DESCRIPTION AND OPERATIONS MANUAL

Release 3.0

(COPYRIGHT © 2003, CSI INTERNATIONAL, INC.)

Override-BUFF is a proprietary product of CSI International, Inc. It cannot be reproduced, changed, copied, or stored in any form (including, but not limited to, copies on magnetic media) without the express prior written.

Original Printing09/24/2003
Last Revised09/24/2003

TABLE OF CONTENTS

Override-BUFF/VSE.....	1
GETTING STARTED.....	1
Introduction	3
What Override-BUFF Does.....	4
Migration Requirements	6
Installation.....	7
Installing Override-BUFF Tape.....	7
USING Override-BUFF.....	9
Override-BUFF Activation and Deactivation.....	11
Override-BUFF Startup Parameters	13
Syntax	13
INIT - Initialization Card	14
EXCLUDE - Exclusion Cards	17
CLASS - Dynamic Partition Class Inclusion Card.....	19
DSNAME - Dataset Inclusion Card.....	20
FILE - File Inclusion Card	21
JOB - Job Inclusion Card	22
PART - Partition Inclusion Card	23
PROGRAM - Program Inclusion Card.....	24
MESSAGES - Message Specification Card.....	25
Parameter Interrelationships	26
Generic Names	28
Parameter Overrides.....	29
Status Report.....	30
Close Statistics	32
Guidelines On Selecting Initiation Values.....	34
CICS Considerations.....	38
How To Deal With Insufficient GETVIS Storage.....	39
A final note on this topic:.....	41
Miscellaneous Notes	42
MESSAGES.....	43
System Console Messages.....	45
BIMVSR-nnn (Routing Facility)	48
APPENDIX A.....	51
A Primer On VSAM Buffering	53
VSAM Buffer Description	53
VSAM Buffer Use Example	53
Estimating Performance Considerations	55
Trade-offs	57
Strings	58
The Surprising Conclusion	59

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 VSE 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 VSE 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" on page 53, 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 DLBL 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 DLBL 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 DLBL statement, irrespective of access type or the availability of storage in different jobs, due to program or partition 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 partition GETVIS 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 performance monitor product.
2. 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, for example when partitions are reallocated. 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 partition GETVIS storage is available, it will be used effectively. If not, additional buffers will not be acquired.
3. Some programs 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.

Two types of files show no improvement under Override-BUFF. They are SAM-managed files and files accessed using user buffers. The latter includes most DL/I files and CICS Auxiliary Temporary Storage. These files will be automatically excluded from modification by Override-BUFF.

Migration Requirements

The following sections cover the changes in the installation and use of Override-BUFF that may be required if you are migrating from an earlier release.

Changes from Release 1.2

- The INIT parameters; FILEMAX, JOBMAX, and PROGMAX have been replaced by a single parameter; ENTRIES. Refer to the section "INIT - Initialization Card" on page 14 for more information.
- The activation and de-activation process has been redesigned. Refer to the section "Override-BUFF Activation and Deactivation" on page 11 for more information.
- A new parameter; DSNAME, has been added to the EXCLUDE card. Refer to page 17 for more information.
- Three new parameters; PARTEXCL, CLASEXCL, and DSNEXCL have been added to the MESSAGES card. Refer to page 25 for more information.
- Three new parameter cards have been added; CLASS, DSNAME, and PART. Refer to pages 19, 20, and 23 for more information.
- New options have been added to the EXEC card to start and stop Override-BUFF, and to reset statistics.

Installation

This release of Override-BUFF/VSE supports VSE/SP releases 2.1 through 4.1, and VSE/ESA 1.1 through 2.7, and will not usually require changes for later releases. Earlier releases of VSE are supported by BIM-BUFF/VSE release 1.1.

Initial installation of Override-BUFF is very easy. (The "real" work comes later when implementing it.) If Override-BUFF is currently installed and you are installing a new version, the old version must be terminated first.

Installing Override-BUFF Tape

Override-BUFF/VSE is distributed on a LIBR BACKUP cartridge. It consists of a sublibrary, named CSIDIST.CSIBUFF. Use a job similar to the following one to restore the sublibrary:

```
* $$ JOB JNM=INSTALL
// JOB      INSTALL
// ASSGN   SYS006, CUU                INSTALL TAPE
// DLBL    USRLIB1   etc.
// EXTENT  etc.
// EXEC    PGM=LIBR, SIZE=256K
//         RESTORE SUBLIB=CSIDIST.CSIBUFF:USRLIB1.CSIBUFF TAPE=SYS006
/*****/
/* Optional step to move members to a permanent */
/* sublibrary and delete sublibrary created above. */
/*****/
CONNECT SUBLIB=USRLIB1.CSIBUFF:USRLIB.PROD
MOVE *.* LIST=YES REPLACE=YES
DELETE SUBLIB=USRLIB1.CSIBUFF
/*****/
/*
/&
* $$ EOJ
```

The following modules and sample JCL will be loaded:

CSIBUFF	System startup/terminate program.
CSIBUFOP	VSAM open interface program.
CSIBUFCL	VSAM close interface program.
CSIBUFPR	Parameter Parser.
CSIBUFTB	Storage anchor for control table.
CSIBUFJ1.A	Sample JCL to load Override-BUFF SVA modules.
CSIBUFJ2.A	Sample JCL to activate Override-BUFF.
CSIBUFJ3.A	Sample JCL to de-activate Override-BUFF.
CSIBUFJ4.A	Sample JCL to reset SDL pointers.
BIMVSRM	Routing Facility command module.
BIMVSRP	Routing Facility Open SVC router.
BIMVSRCL	Routing Facility Close SVC router.
BIMVSRPO	Routing Facility Post Open handler.
BIMVSRPC	Routing Facility Post Close handler.
BIMVSRTB	Routing Facility product table.
BIMVSRJE	Routing Facility end-of-job handler.
BIMVSR0E	Routing Facility SVC 14 intercept.
BIMVSRSC	Routing Facility storage anchor for Close.
BIMVSRSO	Routing Facility storage anchor for Open.

VSAM OPEN/CLOSE Routing Facility

Override-BUFF uses an SVC Routing Facility to intercept VSAM OPEN and CLOSE processing. This facility allows one or more CSI/BIM products to intercept OPEN and/or CLOSE processing in a prescribed order. The Routing Facility is implemented transparently in Override-BUFF.

USING Override-BUFF

Override-BUFF Activation and Deactivation

In the installation library there is a member named CSIBUFJ1.A. Executing this JCL after each IPL will cause the phases listed in the JCL to be loaded into the SVA:

```
// JOB CSIBUFJ1    LOAD SVA PHASES FOR Override-BUFF
/* THIS JOB MUST EXECUTE IN PARTITION BG
// OPTION LOG
// LIBDEF PHASE,SEARCH=USRLIB1.CSIBUFF
SET SDL
BIMVSRQP, SVA
BIMVSRCL, SVA
BIMVSRPO, SVA
BIMVSRPC, SVA
BIMVSRTB, SVA
BIMVSRCP, SVA
BIMVSRJE, SVA
BIMVSR0E, SVA      ('0' IS A ZERO)
CSIBUFTB, SVA
CSIBUFOP, SVA
CSIBUFCL, SVA
/*
/&
```

These phases must reside in the SVA. It is recommended after testing has been completed and Override-BUFF is moved into production, that this member be incorporated into the ASI PROC.

In addition to the SVA modules, the following modules must be in a VSE library that is accessible to all partitions via a permanent LIBDEF:

BIMVSRSC.PHASE
BIMVSRSO.PHASE

This can be accomplished by copying these modules to a VSE library already in a permanent LIBDEF, or by adding the CSIBUFF sublibrary to your permanent LIBDEF.

In the installation library there are also members CSIBUFJ2.A, CSIBUFJ3.A, and CSIBUFJ4.A. The member CSIBUFJ2.A will activate CSIBUFF, and the member CSIBUFJ3.A will stop Override-BUFF.

The member CSIBUFJ4.A will reset all SDL pointers that were modified to bring the system back to its original state.

CSIBUFJ2.A

```
// JOB CSIBUFJ2   START Override-BUFF
// LIBDEF PHASE,SEARCH=USRLIB1.CSIBUFF
// EXEC CSIBUFF
INIT ...
(remaining parameter cards)
/*
/&
```

CSIBUFJ3.A

```
// JOB CSIBUFJ3   STOP Override-BUFF
// LIBDEF PHASE,SEARCH=USRLIB1.CSIBUFF
// EXEC CSIBUFF,PARM='STOP'
/*
/&
```

CSIBUFJ4.A

```
// JOB CSIBUFJ4   RESET SDL/SVA
// LIBDEF PHASE,SEARCH=USRLIB1.CSIBUFF
// EXEC BIMVSRM,PARM='SHUT11'
/*
/&
```

Override-BUFF Startup Parameters

When you start Override-BUFF, you specify a series of parameter cards that describe which ACBs are to be modified. They are specified in a jobstream similar to the following:

```
// JOB CSIBUFJ2   START Override-BUFF
// LIBDEF PHASE,SEARCH=USRLIB1.CSIBUFF
// EXEC CSIBUFF
INIT ...
(remaining parameter cards)
/*
/&
```

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 parameter operands 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 - Initialization 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),  
ACCESS=xxx,BROWSE=xxxxxx,  
GETVIS=(mmmK,nnnK),  
ENTRIES=nnn
```

The INIT parameters are as follows:

ENTRIES=xxx

This parameter specifies the default number of exception entries that are specified in the following cards. Each card counts as at least one entry. Cards that specify multiple exceptions, such as a file card with multiple file names, have a count equal to the number of exceptions specified.

ACCESS=

ACB
RAN
SEQ
BOTH

(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= $\left[\begin{array}{l} (sss,rrr) \\ NO \end{array} \right]$ **(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 or PROGRAM cards as described below.

A value of BUFND=NO may be specified to indicate that only index buffers are to be overridden; this reduces the GETVIS requirements while retaining the buffering advantages for the index component.

BUFNI= $\left[\begin{array}{l} (sss,rrr) \\ NO \end{array} \right]$ **(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 or PROGRAM cards as described below.

A value of BUFNI=NO may be specified to indicate that only data buffers are to be overridden; this reduces the GETVIS requirements while retaining the buffering advantages for the data component.

GETVIS=(mmmK,nnnK) (default value is (24K,48K))

The first value specifies the minimum amount of unused storage Override-BUFF requires in a partition's GETVIS area before it will modify the BUFNI value of an ACB. If this amount of storage is not available, the ACB will not be modified, and the file will be opened using the default (i.e. minimum) buffer values. The second value specifies the minimum amount of unused storage Override-BUFF requires in a partition's GETVIS area before it will modify both the BUFND and BUFNI values of an ACB. If nnnK is not available, but mmmK is available, Override-BUFF will modify the BUFNI value of the ACB (if appropriate) but not the BUFND value. While either of these events will cause degraded performance, this is one tool used in attempt to increase the likelihood that there will be enough GETVIS storage available to open this file and provide for subsequent GETVIS requests.

This parameter is ignored for partitions that have 31-bit GETVIS available.

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 CLASS=(a,b,...z)
EXCLUDE DSNAME=(dataset.name1,...dataset.nameN)
EXCLUDE FILENAME=(filename1,filename2,...filenameN)
EXCLUDE JOB=(aaaaaaaa,bbbbbbbb,...zzzzzzzz)
EXCLUDE PART=(aa,bb,...zz)
EXCLUDE PROGRAM=(aaaaaaaa,bbbbbbbb,...zzzzzzzz)
```

The EXCLUDE parameters are as follows:

CLASS=(a,b,...z)

This EXCLUDE command specifies the classes of dynamic partitions which are never to be modified by Override-BUFF. Only the alphabetic portion should be specified, e.g. specify CLASS=(D,Q) and not CLASS=(D1,Q6)

FILENAME=(filename1,...filenameN)

(default value is no files excluded)

This card specifies the filenames 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 the partition GETVIS space. A filename may be a generic name as described in the section "Generic Names" on page 28.

DSNAME=(dsn1,...dsnN)

(default value is no dsns excluded)

This card specifies the dataset names of datasets which are never to be modified by Override-BUFF. This may be necessary, for example, if a data has a large CI size which would otherwise result in excessive use of the partition GETVIS space. A data set may be a generic name as described in the section "Generic Names" on page 28.

JOB=(job1,...jobN)

(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 section "Generic Names" on page 28.

PART=(aa,bb,...zz)

(default value is no partitions excluded)

This card specifies partitions whose files are never to be modified by Override-BUFF. It is not generally necessary or advisable to exclude CICS, POWER, or VTAM partitions from Override-BUFF processing. This parameter should be needed only under special circumstances.

PART=ALL

This EXCLUDE command specifies that all static and all dynamic partition classes are never to be modified by Override-BUFF. It is merely a convenient technique which may be used instead of specifying each partition and class in individual EXCLUDE statements. This may be useful during trial of the product; individual jobs or partitions may be included by running a separate job step which uses INCLUDE PART=*, which is described in the "Parameter Overrides" section on page 29.

PROGRAM=(pgm1...pgmN)

(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 section "Generic Names" on page 28.

CLASS - Dynamic Partition Class Inclusion Card

The CLASS card may be used to customize the operation of Override-BUFF for a certain VSE Dynamic Partition Class.

```
CLASS class,ACCESS=xxxxx,BROWSE=xxxxx,  
      BUFND=(sss,rrr),BUFNI=(sss,rrr)
```

The CLASS parameters are as follows:

class **(required)**

This specifies the 1 character Dynamic Partition Class.

ACCESS=

ACB
RAN
SEQ
BOTH

This specifies the method which Override-BUFF is to use to determine whether sequential and/or direct processing will be performed. The value, ACCESS=ACB, is appropriate for the vast majority of programs which accurately indicate their intentions in the ACB.

BROWSE=

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 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

This specifies the number of data buffers to be used for all files.

BUFNI=

(sss,rrr)
NO

This specifies the number of index buffers to be used for all files.

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
BOTH]

This specifies the method which Override-BUFF is to use to determine whether sequential and/or direct processing will be performed. The value, ACCESS=ACB, is appropriate for the vast majority of programs which accurately indicate their intentions in the ACB.

BROWSE= [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 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]

This specifies the number of data buffers to be used for all files.

BUFNI= [(sss,rrr)
NO]

This specifies the number of index buffers to be used for all files.

FILE - File Inclusion Card

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

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

The FILE parameters are as follows:

filename **(required)**

This specifies the filename of a file which is to receive special processing by Override-BUFF. The filename may be a generic name as described in the section "Generic Names" on page 28.

ACCESS=

ACB
RAN
SEQ
BOTH

This specifies the method which Override-BUFF is to use to determine whether sequential and/or direct processing will be performed. The value, ACCESS=ACB, is appropriate for the vast majority of programs which accurately indicate their intentions in the ACB.

BROWSE=

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 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

This specifies the number of data buffers to be used for all files.

BUFNI=

(sss,rrr)
NO

This specifies the number of index buffers to be used for all files.

JOB - Job Inclusion Card

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

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

The JOB 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 section "Generic Names" on page 28.

ACCESS=

ACB
RAN
SEQ
BOTH

This specifies the method which Override-BUFF is to use to determine whether sequential and/or direct processing will be performed. The value, ACCESS=ACB, is appropriate for the vast majority of programs which accurately indicate their intentions in the ACB.

BROWSE=

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 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

This specifies the number of data buffers to be used for all files.

BUFNI=

(sss , rrr)
NO

This specifies the number of index buffers to be used for all files.

PART - Partition Inclusion Card

The PART card may be used to customize the operation of Override-BUFF for a VSE static partition.

```
PART partid,ACCESS=xxxxx,BROWSE=xxxxx,  
    BUFND=(sss,rrr),BUFNI=(sss,rrr)
```

The PART parameters are as follows:

partid **(required)**

This specifies the 2 character partition id. The partition id may be a generic name as described in the section "Generic Names" on page 28.

ACCESS=

ACB
RAN
SEQ
BOTH

This specifies the method which Override-BUFF is to use to determine whether sequential and/or direct processing will be performed. The value, ACCESS=ACB, is appropriate for the vast majority of programs which accurately indicate their intentions in the ACB.

BROWSE=

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 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

This specifies the number of data buffers to be used for all files.

BUFNI=

(sss,rrr)
NO

This specifies the number of index buffers to be used for all files.

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)
```

The PROGRAM 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 section "Generic Names" on page 28.

ACCESS=

ACB
RAN
SEQ
BOTH

This specifies the method which Override-BUFF is to use to determine whether sequential and/or direct processing will be performed. The value, ACCESS=ACB, is appropriate for the vast majority of programs which accurately indicate their intentions in the ACB.

BROWSE=

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 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

This specifies the number of data buffers to be used for all files.

BUFNI=

(sss,rrr)
NO

This specifies the number of index buffers to be used for all files.

MESSAGES - 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 (except, perhaps, for the GETVIS message); however, turning some or all of the messages on may help in monitoring the operation of Override-BUFF during the initial testing stages.

The MESSAGES card must appear in your input parameters before any EXCLUDE cards. If more than one MESSAGES card is specified, only the options specified on the last MESSAGES card are used.

```
MESSAGES  BUFND=xxx , BUFNI=xxx , FILEEXCL=xxx ,
           GETVIS=xxx , JOBEXCL=xxx , MODIFIED=xxx ,
           PARTEXCL=xxx , PROGEXCL=xxx , SAMESDS=xxx ,
           USERBUFF=xxx , DSNEXCL=xxx
```

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
GETVIS=YES	ACB not modified because of insufficient GETVIS space
JOBEXCL=YES	ACB not modified because the job was excluded
MODIFIED=YES	ACB was modified
PARTEXCL=YES	ACB not modified because the partition or the partition's class was excluded
PROGEXCL=YES	ACB not modified because the program was excluded
SAMESDS=YES	ACB not modified because the file is a SAM-ESDS(managed SAM) file
USERBUFF=YES	ACB not modified because it specifies that the program will supply its own buffers
DSNEXCL=YES	ACB not modified because the data set was excluded
CLASEXCL=YES	ACB not modified because the class was excluded

Parameter Interrelationships

Due to the diverse sources of buffer values, this section describes the priority by which a file's buffer values are selected, modified or not, and whether a message is produced or not.

The INIT, EXCLUDE, CLASS, DSNAME, FILE, JOB, PART, and PROGRAM cards and operands are processed by Override-BUFF at file open time in the order in which they were submitted on card input.

Any matching EXCLUDE card, found anywhere in the card input, will cause Override-BUFF to not modify the buffer values in an ACB.

If no matching EXCLUDE card is found, then the last values for the ACCESS, BROWSE, BUFND, and BUFNI operands from all other matching cards are used to process a specific ACB. For example, assume the following cards were specified:

```
INIT BUFND=( 5 , 10 ) , BUFNI=( 2 , 3 ) , ACCESS=ACB
JOB ABC , BUFND=NO
JOB DEF , ACCESS=BOTH
PROGRAM IKJ , BUFND=( 1 , 3 ) , BUFNI=NO
EXCLUDE PART=( F3 )
```

All jobs run in static partition F3 will be excluded.

All jobs not run in partition F3, with a jobname ABC, will be processed with values BUFND=NO, BUFNI=(2,3), and ACCESS=ACB except for a step executing program IKJ. That step will use the values BUFND=(1,3), BUFNI=NO, and ACCESS=ACB.

All jobs not run in partition F3, with a jobname DEF, will be processed with values BUFND=(5,10), BUFNI=(2,3), and ACCESS=BOTH except for a step executing program IKJ. That step will use the values BUFND=(1,3), BUFNI=NO, and ACCESS=BOTH.

These processing options can be temporally overridden for specific jobs. Refer to the section "Parameter Overrides" on page 29.

However, regardless of the buffer values determined, an ACB will not be modified by Override-BUFF if any of the following exclusion mechanisms are in effect:

- a. If there is any match by any type of EXCLUDE card or SYSPARM='BUF=EXCL' (except that an EXCLUDE PART can be temporarily overridden by using a SYSPARM='BUF=INCL').
- b. If the component's buffers were specified by the user.
- c. If the ACB is for a SAM-ESDS data set.
- d. If the ACB is for a VSAM catalog.
- e. If the ACB is for an alternate index.
- f. If the GETVIS parameter on the INIT card was exceeded.

The MESSAGES card must appear in your input parameters before any EXCLUDE cards. If more than one MESSAGES card is specified, only the options specified on the last MESSAGES card are 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 GLAHOOG 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 GLAHOOG , BUFND=0 , BUFNI=0 (or EXCLUDE FILENAME=GLAHOOG)
FILE GLA100 , BUFND=0 , BUFNI=20
FILE GLA++++ , BUFND=5 , BUFNI=10
FILE GL+++++ , BUFND=3 , BUFNI=5
```

Parameter Overrides

After Override-BUFF has been initiated, it is possible to include or exclude a partition from Override-BUFF processing in a manner similar to the EXCLUDE card described above. There are two ways of doing this:

- 1) The JCL SYSPARM value.

```
// OPTION SYSPARM='BUF=EXCL'  
// OPTION SYSPARM='BUF=INCL'
```

If BUF=EXCL is specified, the partition in which the OPTION statement is issued will be temporarily excluded from processing by Override-BUFF. If BUF=INCL is specified, the partition in which the OPTION statement is issued will be temporarily included for processing by Override-BUFF (even if excluded by an EXCLUDE PART=xx card when Override-BUFF was initiated). Since VSE resets the value of SYSPARM at end of job, this method is only appropriate for temporary changes. For example:

```
// JOB 3STEPS  
// EXEC SMALPROG  
// OPTION SYSPARM='BUF=EXCL'  
// EXEC BIGHOG  
// OPTION SYSPARM=' '  
// EXEC TYPICAL  
/&
```

- 2) Refresh the Override-BUFF startup parameters using the same JCL used to initiate Override-BUFF. This method will permanently change the Override-BUFF options for all partitions.

Status Report

The status report which Override-BUFF writes to the operator console during termination may also be requested by running a job step similar to the following (Override-BUFF must be active):

```
// EXEC CSIBUFF      (RUN STATUS REPORT)
   STATUS RESET=xxx
/*
           (or)

// EXEC CSIBUFF,PARM='STATUS RESET=xxx'
```

If RESET=YES is specified, the statistical counters are reset to zero. If RESET=NO is specified, the statistical counters are not reset. The default value is RESET=NO. The statistics are generated as messages CSIBUFF-119 and CSIBUFF-120, and include the following:

- The number of direct (random) ACBs which have been modified
- The number of sequential ACBs which have been modified
- The number of ACBs which have not been modified because:
 - a. The file was excluded because it was specified on an EXCLUDE card or BUFND=0 or BUFNI=0 was specified on the INIT or a FILE card.
 - b. There was insufficient GETVIS storage according to either of the GETVIS values on the INIT card.
 - c. The job was excluded because it was specified on an EXCLUDE card or BUFND=NO was specified on a JOB card.
 - d. The program was excluded because it was specified on an EXCLUDE card or BUFND=NO was specified on a PROGRAM card.
 - e. The partition was excluded because it was specified on an EXCLUDE card, or an OPTION SYSPARM='BUF=EXCL' card was in effect.
 - f. The program specified BUFND and/or BUFNI values in the ACB which were other than the default values.

Since Override-BUFF can handle the BUFND and BUFNI values separately, a file which is opened for both random and sequential access may be counted twice in the above statistics, once corresponding to the BUFND value and once corresponding to the BUFNI value. It must not be assumed that the sum total of all these statistics is the number of VSAM opens which have been processed.

Close Statistics

Override-BUFF contains a parameter called CSTAT which may be used to show the effect which 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 modifying ACBs.

When the CSTAT option is used, several messages are written to the operator console each time a VSAM file is closed. These messages (CSIBUFF-121, CSIBUFF-122, and CSIBUFF-123), indicate the number of logical requests issued by the program to VSAM and the number of physical I/Os that actually took place.

This corresponds to similar statistics maintained in the VSAM catalog, the difference being in the time frame involved. Override-BUFF's statistics reflect the activity of one execution of a program between the time the file is opened and the time it is closed; LISTCAT's statistics reflect the total activity for all processing since the file was DEFINED. CSTAT is especially useful when performing benchmarks to determine the effectiveness of Override-BUFF.

Override-BUFF also checks the buffer space in use for the file. If too much buffer space is used for either the data or index component, a message is issued. While this is not necessarily a problem, it may indicate that the Control Interval size should be reduced or that fewer buffers should be allocated.

CSTAT processing may be initiated and terminated at any time and from any partition. Run a job step similar to the following:

```
// EXEC CSIBUFF      (CLOSE STATISTICS INITITATE/TERMINATE)
      CSTAT parameters
/*
```

The CSTAT parameter can also be specified with your startup parameters if you want the statistics to be active when Override-BUFF starts.

The format of the CSTAT card is as follows:

```
CSTAT INIT, BUFSPD=mmmK, BUFSPI=nnnK
CSTAT TERM
```

The CSTAT parameters are as follows:

INIT indicates that CSTAT message's are to be initiated

TERM indicates that CSTAT message's are to be terminated

BUFSPD=mmmK (default is 30K)

If the data component used more than this amount of storage for buffers, message CSIBUFF-123 is to be issued. A value of 0K may be used to force a message every time a file is closed.

BUFSPI=nnnK (default is 12K)

If the index component used more than this amount of storage for buffers, message CSIBUFF-123 is to be issued. A value of 0K may be used to force a message every time a file is closed.

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 partition GETVIS storage.

This is covered in detail in the section "How To Deal With Insufficient GETVIS Storage" on page 39. (By the way, GETVIS 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 GETVIS 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 partition GETVIS 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
- For operational convenience, be prepared to modify jobs to use OPTION SYSPARM="BUF=EXCL" and/or "EXCLUDE PART=*" on an emergency basis if needed until a permanent resolution can be implemented.
- The number of files, jobs, partitions 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, CLASS, DSNAME, FILE, JOB, PART, 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, CLASS, DSNAME, FILE, JOB, PART, 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 GETVIS 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 partitions active simultaneously, it should be possible to minimize GETVIS 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.
- The purpose of the GETVIS parameter on the INIT card is to provide a "soft landing" as partition GETVIS storage nears filling up. This is a distinct advantage over IBM's use of the BUFFERSPACE parameter in the VSAM catalog, ACB, or DLBL statement. If BUFFERSPACE is used, the value specified becomes an absolute requirement, and the open will fail if the specified amount of storage is not available. However, if the unused partition GETVIS space is in danger of exhaustion, Override-BUFF is able to attempt to open the file with the default BUFND and/or BUFNI values; while this is less efficient, at least the program stands a better chance of being able to run. Generally, it is best to specify the GETVIS parameters on the INIT card as low as possible since this attempts to make the most use of the partition GETVIS storage. Then, the files and programs which tend to be "hogs" may be specified specially. If it turns out that there are a large number of these "hogs", an increase in the GETVIS parameters may be convenient. An alternative to raising both of the GETVIS values would be to raise just the second. For the reasons discussed above, this will tend to allow the efficiencies of multiple index buffers while reducing the storage requirements of multiple data buffers.
- 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, CLASS, DSNAME, FILE, JOB, PART, 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 CLASS, DSNAME, FILE, JOB, PART, and PROGRAM cards are to be preferred to the EXCLUDE card.
- The DSNAME, FILE 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 partition: 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. There is no measurable performance gain if the CICS partition is excluded from Override-BUFF's list of partitions on the EXCLUDE PART= card when Override-BUFF is initiated. If an installation has files which use the default buffer specifications, Override-BUFF may certainly be used to quickly increase those values. However, if there are many such files, there may be insufficient GETVIS space to open them all at the same time. In this case, the CICS partition can be excluded from Override-BUFF processing. 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 partition 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.

How To Deal With Insufficient GETVIS Storage

Most installations will discover that most jobs run normally (but faster) after Override-BUFF has been initiated. However, there may be a few jobs which will cancel due to insufficient GETVIS storage. This will probably be accompanied by message "4228I FILE xxxxxxxx OPEN ERROR X'88'(136)" or "4228I FILE xxxxxxxx OPEN ERROR X'B4'(180)". The actual number of jobs affected will vary for each installation. Most installations have allocated more partition GETVIS storage than is necessary, thus allowing Override-BUFF to utilize this unused storage. Installations which have allocated GETVIS storage very precisely may find that those allocations are sometimes insufficient when Override-BUFF is running.

Fortunately, this will probably be limited to a few programs which use a large number of files or which access a file having a very large CI size. Generally, GETVIS storage is an unmanaged commodity in most installations. As long as there is enough storage so jobs don't cancel, no one bothers about how much is really needed or whether it is being used efficiently. This lack of information can make it difficult to deal with the problem of insufficient GETVIS space. However, there are a number of ways of dealing with this situation, and the problem should, by no means, be insurmountable.

- The simplest and most immediate approach is to increase the partition's GETVIS storage by decreasing the SIZE parameter on the EXEC card. As long as this leaves enough storage below the SIZE value for the program(s) to operate, this approach is entirely satisfactory. Note that this approach may be needed only for a relatively small percentage of the jobs in a given partition.
- If the above approach proves sufficient but involves a large number of JCL changes, the same affect may be achieved by decreasing the value(s) specified on the SIZE statement (not the ALLOC) statement which is usually executed in the BG ASI procedure.
- A similar approach is to increase the size of the partition(s) experiencing insufficient GETVIS storage while leaving the SIZE parameters unchanged. The difficulty with this approach occurs when all 16M bytes of storage in one address space are currently allocated. In this case, adding to one partition must necessarily reduce the storage available to another partition. It is probably true that most installations could obtain an additional 512K to 2M of virtual storage without harming any partition by utilizing storage more efficiently. Some possible candidates are:

- a. The BG partition must be at least 900K in order to run MSHP; however, many installations run with BG much larger. This can probably be safely lowered to 900K (but no less).
 - b. Some installations have defined all 12 possible partitions, one or more of which are rarely, if ever, used. These might easily be removed without impacting the usefulness of the system.
 - c. The SVA at a typical installation is larger than it needs to be for two reasons. First, it contains space which is allocated but never used. Second, it contains many programs which are loaded to the SVA but are never used or are used only infrequently. Potential savings are 100K to 1M.
 - d. The system GETVIS area is typically allocated more storage than is needed. Potential savings are 50K-300K. Be careful, however, since insufficient system GETVIS may cause an important software product to fail. If so, an IPL will probably be necessary in order to increase its size again.
 - e. CICS, due to its complexity, is often given too much virtual storage. The difficulty here lies in knowing where it can be reduced. Possible candidates are reducing the number of resident programs, decreasing the size of the Dynamic System Area (DSA) and making more efficient use of VSAM buffers. Potential savings are 200K to 2M. Again, care must be exercised to insure that this is not accomplished by preventing certain processes from being performed or by reducing the performance of the CICS system.
 - f. ICCF users often have more interactive partitions than are actually needed. Furthermore, they are often larger than are actually needed. For most uses, a few 128K interactive partitions should prove sufficient. Potential savings are 200K to 1M.
- If it is difficult to make additional partition GETVIS storage available, Override-BUFF has a number of options which can be used to utilize the available storage more effectively.
 - a. The most immediate solution is to use `// OPTION SYSPARM='BUF=EXCL'` which will cause Override-BUFF to allow the default (i.e. minimum) buffer values to be used for all files in the affected partition. Since VSE resets the SYSPARM value at end of job, this may be suitable for temporary overrides for certain problem jobs.
 - b. When initiated, Override-BUFF may be given EXCLUDE cards which specify that the default (i.e. minimum) buffer values are to be used for certain files, jobs, programs or partitions. These values remain in effect as long as Override-BUFF is initiated.

- c. When initiated, Override-BUFF may be given FILE cards which specify that smaller buffer values (but presumably larger than the minimum) are to be used for certain files. These values remain in effect as long as Override-BUFF is initiated.
 - d. When initiated, Override-BUFF is given GETVIS parameters which indicate at what point Override-BUFF is to allow the default buffer values to be used. Raising these values will cause Override-BUFF to allow for a larger "cushion" as the GETVIS nears its capacity. On the other hand, lowering these values will allow the maximum amount of GETVIS storage to be utilized.
 - e. When initiated, Override-BUFF is given the BUFND and BUFNI values which are to be used to override the default values. These values may be lowered to use less GETVIS storage, but at the expense of increased I/Os and CPU usage. Since the greatest performance benefit from the increased buffering provided by Override-BUFF comes from the use of additional index buffers, but the greatest increase in storage requirements comes from data buffers, one possible solution would be to use BUFND=0, which will cause the minimum number of data buffers to be used at all times.
- If one or more files have a large CI size (perhaps 8K or more) and are usually accessed randomly, it may be desirable to actually reDEFINE the file so it uses a smaller CI size. Then it can be processed by Override-BUFF instead of being specifically mentioned on an EXCLUDE or FILE card.
 - Run in Virtual Addressing Extended (VAE) mode which allows for more virtual storage to be allocated to partitions than running only a single address space.

A final note on this topic:

If Override-BUFF produces the benefits claimed in this manual, it is probably worth some effort to resolve the problem of insufficient partition GETVIS storage. While the above discussion may seem to be rather extended and unnecessarily complex, one reason for this is that there are so many ways to resolve the issue. Pursuing the approaches which seem the easiest to use should produce the desired results within a reasonable amount of time. Also, do not consider GETVIS problems to be inevitable; most installations do not experience GETVIS problems after installing Override-BUFF.

Miscellaneous Notes

- Due to the way the VSAM Space Management for SAM Feature is implemented, sequential files which are managed by it will not benefit from allocating additional buffers; therefore Override-BUFF will not modify their ACBs.
- 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.

- DL/I supplies its own buffers instead of using VSAM's buffers. Override-BUFF will recognize this and automatically exclude DL/I files. There is one exception; a HIDAM KSDS uses VSAM buffers, and Override-BUFF will modify the ACB if all of the standard conditions are met.
- 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 GETVIS 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.

MESSAGES

System Console Messages

The following messages may appear on the job log:

CSIBUFF-100	<table border="1"><tr><td>BUFND BUFNI</td><td>FOR dataset.name</td><td>MODIFIED TO n SPECIFIED NO IN PARMS HAD ACB BUFFERS HAD BUFFS IN PGM</td></tr></table>	BUFND BUFNI	FOR dataset.name	MODIFIED TO n SPECIFIED NO IN PARMS HAD ACB BUFFERS HAD BUFFS IN PGM
BUFND BUFNI	FOR dataset.name	MODIFIED TO n SPECIFIED NO IN PARMS HAD ACB BUFFERS HAD BUFFS IN PGM		
	<p>One or more of these messages is displayed depending on the MESSAGES card operands that have been specified.</p>			
CSIBUFF-101	<p>CSIBUFTB TABLE NOT FOUND Override-BUFF is not currently running.</p>			
CSIBUFF-103	<p>GETVIS ERROR, (error description) Override-BUFF cannot be initiated due to a failure of its GETVIS macro.</p>			
CSIBUFF-104	<p>SYSIN ERROR, (error description) An invalid or misspelled parameter has been specified.</p>			
CSIBUFF-105	<p>NO PARMS SPECIFIED ON EXECUTE CARD You must specify one of the following parms on the EXEC statement: STRT, STOP, or REFR.</p>			
CSIBUFF-106	<p>INVALID PARM SPECIFIED An invalid or misspelled keyword has been specified. The keyword is the first character string on the card, such as "INIT"</p>			
CSIBUFF-107	<p>FREEVIS FAILED, (error description) A FREEVIS macro issued by Override-BUFF has failed. This should never occur.</p>			
CSIBUFF-108	<p>TABLE NOT REFRESH, NOT LOADED Override-BUFF tried to refresh the CSIBUFTB table, but it was not loaded. Use the PARM=STRT option to activate Override-BUFF.</p>			
CSIBUFF-112	<p>ERROR RETURNED FROM ROUTER xxxx Request technical assistance for this error.</p>			
CSIBUFF-114	<p>MODULE xxxxxxxx NOT FOUND Required module 'xxxxxxx' could not be loaded. Refer section "Installation" on page 7 for proper load library contents and setup.</p>			

CSIBUFF-115 ERROR RETURN FROM CSIBUFPR

Request technical assistance for this error.

CSIBUFF-116 Override-BUFF RELEASE 2.0x SUCCESSFULLY xxxxxxxx

The specified version of Override-BUFF has been successfully initiated, stopped or refreshed.

CSIBUFF-117 LABEL MACRO ERROR RC = xxxxxxxx ACB NOT MODIFIED (OP)

Request technical assistance for this error.

CSIBUFF-118 EXCLUDE FOR dataset.name PER

PGM	xxxxxx
DSN	
JOBNAME	
FILENAME	
PART	
CLASS	

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

CSIBUFF-119 nnn DIRECT AND nnn SEQ ACBS WERE MODIFIED

CSIBUFF-119 nnn ACBS NOT MODIFIED DUE TO

BUFND NOT ZERO
BUFNI NOT ZERO
FILE EXCLUDE
GETVIS EXCLUDE
JOB EXCLUDE
PARTITION EXCLUDE
PROGRAM EXCLUDE
SAM-ESDS EXCLUDE
USER BUFFERING
DSN EXCLUDE
CLASS EXCLUDE

One or more of the above messages are displayed when the STATUS command is executed.

CSIBUFF-120 STATS FIELDS RESET

Informational message to confirm that a STATUS job was executed with the CLEAR=YES operand.

CSIBUFF-121 (file) DSN dataset.name

This message is generated when close statistics (CSTAT) is active.

**CSIBUFF-122 (file)

DATA
INDEX

 EXCP=nnnnn I/O= nnnnn**

These messages are generated when close statistics (CSTAT) are active. We have determined that the I/O count for INDEX reads is not maintained by VSAM.

- CSIBUFF-123** (file)

DATA
INDEX

BUFFERS=nn SPACE= nnnnnK CISIZE=nnnnn
These messages are generated when close statistics (CSTAT) are active and the bufferspace used exceeds the specified limits.
- CSIBUFF-124** **CSTAT REQUEST PROCESSED**
Informational message to confirm that the CSAT parameter has been successfully processed.
- CSIBUFF-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.
- CSIBUFF-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.

BIMVSR-nnn (Routing Facility)

The following messages are generated by the VSAM Routing Facility used by several BIM/CSI software products.

BIMVSR-001	xxxxxxx MODULE NOT FOUND The module "xxxxxxx" could not be found by the Router. Make sure that the library containing program "xxxxxxx" is accessible to all partitions.
BIMVSR-003	NO INIT PARMS (error description) Request technical assistance for this message.
BIMVSR-005	TABLE xxxxxxx NOT FOUND The table "xxxxxxx" could not be found by the Router. Make sure that the library containing table "xxxxxxx" is accessible to all partitions.
BIMVSR-006	BIMVSR NOT ACTIVE Request technical assistance for this message.
BIMVSR-009	MODULE xxxxxxx FOR MOD yyyyyyy SUBROUTINE ERROR Request technical assistance for this message.
BIMVSR-010	GETMAIN ERROR (error description) Request technical assistance for this message.
BIMVSR-011	ROUTER ALREADY ACTIVE The router has already been activated.
BIMVSR-015	FREEVIS ERROR (error description) Request technical assistance for this message.
BIMVSR-016	NO ENTRY IN TABLE BIMVSRTB FOR APPL Request technical assistance for this message.
BIMVSR-017	CDLOAD FOR MODULE xxxxxxx FAILED Request technical assistance for this message.
BIMVSR-018	MODULE xxxxxxx NOT FOUND IN SVA, RC=x Request technical assistance for this message.
BIMVSR-019	PRODUCT xxxxxxx NOT FOUND Request technical assistance for this message.

- BIMVSR-020** **ERROR IN SRCSDL ROUTINE xxxxxxxx PART, SDSDSRC=x**
Request technical assistance for this message.
- BIMVSR-021** **THE SVC TABLE COULD NOT BE FOUND**
Request technical assistance for this message.
- BIMVSR-022** **WACOMMON STORAGE NOT FOUND**
Request technical assistance for this message.

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, DOS/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 the partition's GETVIS area. The GETVIS area is the storage which is above that specified on the EXEC card and below the size of the partition. The GETVIS area is 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 buffers * 1024 bytes = 3072 + 300 = 3372 bytes

data: 2 buffers * 4096 bytes = 8192 + 200 = 8392 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 partition size of 256K to 768K, 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. The time required to process one record and print one print line (under POWER) 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	5 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 partitions, 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 partition's GETVIS area. While it sounds like a simple idea to merely add more GETVIS 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. This problem is discussed in detail in the section "How To Deal With Insufficient GETVIS Storage" on page 39.

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 on-line 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 BUFSP on the DLBL statement or BUFFERSPACE when DEFINEing a file, this method has a number of disadvantages:
 - a. They do not allow individual control of index (BUFNI) and data (BUFND) buffers.
 - b. They become fixed requirements; if the specified storage is unavailable, the job will be canceled.
 - c. If the BUFSP parameter on the DLBL statement is to be modified, this involves making changes to JCL.
 - d. 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.