CA Hyper-Buf® VSAM Buffer Optimizer

User Guide

r11.5

CA Technologies Product References

This document references the following CA Technologies products:

- CA Hyper-Buf® VSAM Buffer Optimizer (CA Hyper-Buf)

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Introduction

CA Hyper-Buf improves VSAM buffer space allocation to reduce response time, elapsed job time, and I/O rate. This manual serves as a guide to the buffering procedures and facilities used by VSAM to accelerate processing and provide maximum performance.

CA Hyper-Buf chooses proper buffer sizes by dynamically analyzing the data set environment. CA Hyper-Buf computes buffer spaces based on the following factors:

- Hardware construction
- DASD type and track capacity
- Available virtual storage
- File access pattern
- File definition parameters
- File sizes
- Data and index CI sizes
- User customization parameters

Dynamic buffering is typically ineffective or impractical for jobs like CICS that open multiple files. What works for batch can keep CICS from coming up or otherwise negatively affect CICS performance. CA Hyper-Buf has special algorithms for tuning CICS buffer space. By coding one rule for any CICS partition, you can tune CICS to select optimum buffer space for online processing.

This section contains the following topics:

Dynamic VSAM Buffering

Buffering affects VSAM performance. Optimum buffering leads to improved performance, which results in shorter online response times, increased batch throughput, and shorter batch turnaround times. Improper buffering causes job backlogs, missed deadlines, and irate terminal operators.

This chapter presents an overview of the buffering techniques used by VSAM and explains how proper buffering accelerates VSAM processing.

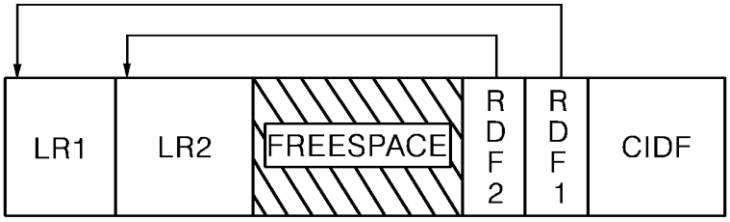
VSAM Buffering and Performance

VSAM manages disk accesses on behalf of application programs. To do this, VSAM reserves blocks of virtual storage called buffers, which are used as staging areas for data that is on its way to or from disk.

Buffering Basics

The basic unit of data transfer between DASD and a buffer is a control interval (CI). At least one CI is read from or written to disk with each physical I/O operation. VSAM never transfers less than one CI. In certain instances, VSAM can transfer several CIs with a single I/O request by chaining several CCWs together into a single channel program. By reading or writing several CIs with a single I/O, fewer I/Os are required to process the data. This reduction in I/O results in improved performance.

Each CI usually contains more than one logical record. In this respect, CIs are similar to disk blocks that are used by non-VSAM access methods. Some CIs can contain hundreds of logical records, others can contain only a few records. Still other CIs can contain a single logical record, or even a portion of a logical record in the case of spanned records. A diagram of a control interval and its internal structure is shown below.



VSAM Disk Data Structures

In the VSAM Control Interval diagram, the fields marked LRx represent Logical Records contained in the control interval. The fields marked RDFx represent Record Definition Fields. The last field, marked CIDF, represents the Control Interval Definition Field. The shaded area marked FREESPACE is space that is currently unused in this control interval.

The RDFs and CIDF are VSAM control fields that are contained in almost every control interval. These fields are wasted space in the sense that they contain no useful application data. However, these fields contain critical VSAM information, and are required.

Note: RDFs exist only in CIs that contain logical records. An empty CI does not contain any RDFs. Even empty CIs contain a CIDF, except CIs within a Linear Data Set (LDS). LDS clusters contain only application data; there are no CIDFs or RDFs in any LDS CI.

VSAM Control Intervals and Buffering

KSDS files contain two types of control intervals. The first type of control interval is the data control interval, which contains the logical records and VSAM control information as shown in the VSAM Control Interval diagram in the section Buffering Basics (see page 10). The other type of CI is the index control interval. The index CIs contain index entries which point to other index CIs or to data CIs. These indexes allow VSAM to retrieve records randomly by a field in the record which is called a key. VSAM maintains index CIs transparently on behalf of the application programs.

All data CIs for a particular cluster are the same size. All index CIs for a particular cluster are also the same size, although the data control intervals for a particular cluster can be a different size than the index control intervals for the same cluster. For example, a particular KSDS can be defined with 4K data CIs and 1K index CIs.

Each virtual storage buffer is the same size as the control interval for which it is built. For example, a KSDS file with an index CI size of 1K and a data CI size of 4K requires some number of 1K buffers for index control interval I/O, and some number of 4K buffers to be used exclusively for data CI I/O.

Application Interface

The application program is usually interested in logical records and not control intervals. VSAM manages the control intervals on behalf of the application by:

- Finding the control interval that contains the requested logical record. For keyed requests against a KSDS file, VSAM first searches the index. This processing usually requires at least one I/O operation, and sometimes multiple I/Os. These I/Os use the buffers that were allocated for the index CIs.
- Retrieving the data control interval. The preceding step found where the record resides; now VSAM must actually read that data CI. This processing usually requires at least one I/O operation, and can require several I/Os in the case of spanned records. These I/Os use the data buffers.
- Deblocking the requested logical record. Applications are usually written to handle a single logical record at a time - not an entire CI. Once VSAM reads the entire data CI into a buffer, VSAM then finds the desired logical record and extracts just that record for the application.
- Passing the logical record back to the application or inserting the logical record into the control interval for a write request. Once the record is located, it is usually moved to an application program work area (move mode processing).

Note: The first and second items can require several I/Os to the DASD. Some of these I/Os can be avoided if the required control intervals are already resident in virtual storage buffers.

Extra Buffers

Increasing the number of buffers available to VSAM increases the probability of finding data in a buffer. Records already in a buffer are usually accessed without requiring any I/O, meaning fewer I/Os and I/O waits, shorter elapsed times, reduced I/O path contention, and faster response times for online applications.

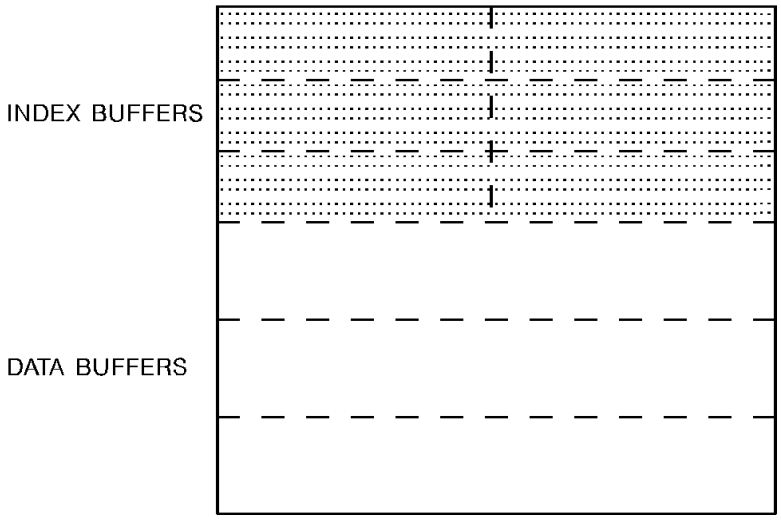
Non-Shared Resources (NSR)

With NSR buffering, VSAM acquires a private pool of buffers for each file when it is opened. The storage allocated to these buffers is reserved for I/O from/to this file for the entire time that the file is open. At CLOSE time, these buffers are released back to the system.

VSAM provides parameters which allow you to specify the number of buffers assigned to a cluster. The VSAM buffer parameters are BUFFERSPACE, BUFND, and BUFNI.

BUFFERSPACE

The BUFFERSPACE parameter specifies the number of bytes of virtual storage reserved for buffers when the file is opened. VSAM open processing GETMAINs BUFFERSPACE bytes of storage. VSAM then subdivides this block of storage into individual data buffers and, if the file is a KSDS, index buffers. The following diagram shows BUFFERSPACE bytes of storage subdivided into six index and three data buffers.



For KSDS files using NSR buffering, VSAM subdivides the BUFFERSPACE area into some number of buffers for data control intervals, and some number for index control intervals. The actual number of each type depends upon the number of bytes available for buffers (BUFFERSPACE), the ACB MACRF options specified at OPEN time (DIR, SEQ, or both), and the CI sizes of the data and index components.

Specifying BUFFERSPACE

You can specify BUFFERSPACE in three places.

- By IDCAMS BUFFERSPACE when you define a cluster
- On a DD card, using the AMP parameter
- In the ACB that is used to access the cluster

Note: The first method assigns buffer space prior to the actual use of the file at the time of the cluster definition. The other two methods assign buffer space when the cluster is actually opened.

Problems Related to Static Buffering

Specifying BUFFERSPACE at DEFINE time, or BUFFERSPACE, BUFND, or BUFNI at execution time using the AMP DD parameter or the ACB, is a static form of buffering. It is static because the values are not dynamically adjusted for any changes in the environment.

As a file grows, or as new applications process the cluster, or as modifications are made to the types of accesses made by existing applications, the numbers that were specified remain as originally assigned. You must monitor and adjust these parameters manually as the environment changes if optimum performance is desired. The only way that you can adjust these parameters in a dynamic production environment is by continuously monitoring each and every program and cluster. Whenever anything changes, you readjust the number of buffers manually.

Static buffering allows you to tune VSAM for good performance. If you can determine how a particular application accesses a particular file, and then determine an optimum number of data and index buffers for this application to process this cluster. You can tune VSAM for the best possible performance, as long as nothing ever changes.

Examples of Static Buffering:

For example, if an application processes a KSDS file randomly, extra index buffers can reduce the number of I/Os required to search the index component. Buffering the entire index set avoids all read I/Os to the index set, except the first I/O for each index set control interval. Buffering the index set significantly improves the performance of this random application (faster online response times, reduced I/O path congestion, shorter elapsed time for batch, and so forth.)

Assume that a KSDS file has three index levels and that an application accessing this KSDS is currently using the default buffer space (two data buffers, one index buffer) to randomly retrieve logical records. Each retrieval requires at least four I/Os to the disk.

A random access request to a KSDS file requires VSAM to search the index structure to find the key of the requested logical record. VSAM always starts this search with the high level index record. If the high level index record is not already available in virtual storage, VSAM must perform one I/O to read it into an index buffer (first I/O). Since this example assumes a three level index, the first matching-or-greater key that is encountered in the high level index record points to another index record in the middle level of the index.

VSAM must next search this middle level index control interval. This requires an I/O (second I/O), since this second level index control interval is not already in a buffer.

Since we are using the default buffer space in this example, we have only one index buffer, which VSAM filled with the high level index record on the previous I/O. So VSAM must read the middle level index control interval into the only index buffer over the top of the high level index control interval currently resident there. The first matching-or-greater key that VSAM finds in this middle control interval points to a bottom level index control interval, called a sequence set control interval, which must now be read into the only index buffer (third I/O) overlaying the middle level record that is currently in residence.

The first matching-or-greater key encountered in the sequence set record points to the data control interval that contains the logical record that was requested by the application (if it exists at all). This data control interval must now be read into a data control interval buffer which requires at least one I/O, and possibly more if the data record is SPANNED (fourth and subsequent I/Os).

This same process is repeated for each logical record that is accessed randomly by this application. The next random request requires the high level index CI, which causes an I/O, since the copy of the high level index record (obtained by the first I/O) has been overlaid by the subsequent index I/Os, and is no longer resident.

This means that:

- An application waits for at least four I/Os to complete for each logical record accessed. This includes the possibility of up to four waits for a busy I/O path (channel/controller/device), four seeks, four device latencies, and four data transfers, and includes all of the processor cycles required to drive 4 I/Os and to process four I/O interrupts (control block validation, page fixing, channel program construction, SVC overhead, interrupt handler, pagefree, SRBs, and so forth). Online response time and batch throughput suffers because of this additional I/O and processing. This application tends to be I/O bound, and spends most of its time waiting for I/Os to complete, instead of performing the desired processing of the data!

- The disk(s) on which this cluster is defined is busy performing up to four I/Os for each logical record accessed. Device congestion slows down not only the application causing the excessive I/O, but all applications using files on this disk.
- The controller, head of string, and channel are also busy, adding to overall system performance degradation. The I/O subsystem (IOCP) can be forced to delay I/O requests by queuing or suspending them until a path or device is available.

Increasing the number of index buffers reduces the number of I/Os required for each randomly accessed logical record. VSAM finds more of the index control intervals in a buffer and avoids index set I/Os. To reduce the number of index I/Os to a minimum requires as many index buffers as there are index set records, plus one per string for sequence set record buffering.

To buffer the entire index set manually, you must first determine how many index set control intervals exist in the index component, then specify that many index buffers plus one per string in the BUFNI parameter through the ACB, or on the DD card in the AMP parameter. This specification allows

VSAM to keep an 'index set cache' in main storage and to avoid all of the read I/Os to the index set (except the first I/O for each control interval which is required to make that CI resident in a buffer). This improves online response times, reduces path congestion, and reduces batch turnaround times for applications that access this file randomly.

You can specify buffers using the BUFFERSPACE, BUFNI, and BUFND parameters. You can specify some of these parameters at DEFINE time (BUFFERSPACE only), at execution time using the ACB, or on the DD card in the AMP parameter.

General Purpose Buffers

It is difficult, if not impossible, to select a buffer space at DEFINE time that optimizes the performance of the wide range of applications and access patterns that are found in a production environment. Most files have several applications accessing them, including sequential backups and reorgs, audits, random online inquiries, and batch reports. Each of these applications has a different accessing pattern, which requires buffering adjustments to perform optimally. While specifying a larger buffer space in the catalog usually improves performance, it does not result in the best possible performance for each application that accesses a file. Some applications are better served by the catalogued buffer space than others. Also, the specified buffer space can waste virtual storage, possibly causing VSAM to allocate more index buffers than are required to buffer the entire index set. These excess buffers are not useful, and the virtual storage thus tied up could be put to better use elsewhere.

Specifying BUFFERSPACE at OPEN Time

To optimize performance, you must specify buffer space at OPEN time.

This allows you to tailor the buffer space to suit each particular application and to optimize VSAM performance for the processing that each application performs.

If you calculate that ten index buffers are required to hold the entire index set in storage for a particular file (assume one high level index control interval, eight middle levels, and one sequence set buffer for single string access), you can then modify each and every job stream that accesses this file randomly for optimum performance. Simply specify BUFNI=10 in the AMP parameter on every DD card in every job (test and production) that references this file. These modifications take some time if there are many jobs that access this file, but the end result is optimum performance.

This is static tuning. It works, but you must monitor all of your VSAM files and applications regularly to be sure that performance is optimized. What happens to performance when the file grows to 12 index set records and four index levels? You must go back and adjust all of the BUFNI and BUFND values to reflect the current environment.

Dynamic Environments

The static tuning performed above falls apart as the file grows. As the file grows, the number of index records increases along with the increase in data. The specification of 10 index buffers no longer optimizes the buffering for this file because of the growth. Performance gradually degrades as applications require more and more index I/Os to randomly retrieve logical records. Performance with BUFNI=10 is better than with the default buffer space. These extra buffers are put to good use, but performance is no longer optimum.

To optimize performance after such changes occur, you must recalculate the number of index set records and update all of the DD statements in all of the job streams to include the new optimum BUFNI value. This procedure takes some time, but the result is good performance.

Similar performance problems occur when an application is modified to perform skip sequential or purely sequential processing instead of just random processing. You must monitor and adjust buffers to keep up with the changes that occur to both files and applications.

For sequentially processed files, extra data buffers are beneficial, and extra index buffers generally do little to improve performance. For sequential applications that add a significant number of logical records to a file, extra index buffers may improve performance. As records are added sequentially to the file, VSAM must update the index as CIs are added or split. Since changing one level of the index can require changes to other higher levels, VSAM may require access to several index CIs to keep the index updated. In this case, I/Os can be reduced by adding extra index buffers.

You can decide to set BUFND to a value that allows VSAM to perform full track I/O, thus reducing the number of I/Os and the elapsed time required to process this file sequentially.

What happens if the cluster is moved to another device type or, again, if the application is modified and now performs skip sequential (or maybe even truly random) processing? The answer is simple: recalculate the BUFND, and possibly BUFNI, and modify every job stream (yet again) to specify the new values.

Static buffering, while better than default buffering, is a far from perfect solution in a dynamic, real world environment.

Dynamic Buffering

CA Hyper-Buf determines the optimum number of buffers for a particular file and application accessing that file at the last possible moment, at OPEN time. CA Hyper-Buf inspects the file and determines the number of index levels, number of index set records, data control intervals per track, and the options specified for this particular OPEN (random, sequential, skip sequential). CA Hyper-Buf then allocates the optimum number of buffers, within constraints established by your installation. Each application has the number of buffers optimized for that application using that file, in whatever state the file happens to be at the time of the OPEN.

CA Hyper-Buf is automatic and requires no changes to your application programs, DD statements, or catalogs. As your files or applications change, CA Hyper-Buf automatically adjusts buffer space for optimum performance. The changes are transparent in every way, with the exception of improved online response times and higher batch throughput.

CA Hyper-Buf is dynamic buffering instead of static buffering. Values are adjusted automatically as the environment changes, all transparently to the applications and users.

Local Shared Resources (LSR)

With LSR buffering, several aspects of VSAM processing are different than with NSR buffering. LSR buffers are tuned for random access, and can dramatically reduce I/Os for random access applications.

As you have seen, NSR buffering can reduce I/Os for sequential processing by transferring multiple CIs with a single I/O operation, effectively pre-staging data for the sequential process. Since the sequential process (predictably) requires the read-ahead data, performance is enhanced.

There is no pre-staging of data when LSR buffers are used. Therefore, applications that perform significant amounts of sequential processing should avoid using LSR buffers. Since there is no read-ahead with LSR, sequential applications generally require more I/Os with LSR than with NSR buffers.

For random applications, LSR can dramatically improve performance over NSR buffering. With NSR, you can expect to perform at least 1 index I/O (for the sequence set) and 1 data I/O for each random access request. With LSR, you can effectively avoid all index I/O, after the first I/O for each CI to make it resident - an effective I/O reduction of 50 percent.

Additionally, with LSR you can defer writes for CIs that can be updated multiple times within a short period of time (with NSR, each CI is written back to disk every time data in the CI is altered; therefore, the in-storage copy of a single CI can be written to disk several times when a transaction changes fields in several records that reside in the CI).

The LSR buffers act as a main storage cache of records, both index and data. Records that are referenced frequently tend to remain in storage, and the application will avoid I/O for each record that is found in this cache.

To further reduce I/O, relatively large quantities of LSR data can be staged in hiperspace. Data in a hiperspace can be accessed synchronously without any paging, I/O or interrupt overhead.

JCL Requirements

Guidelines to assist you in preparing your JCL are provided in this manual. The sample code provided in this document is intended for use as a reference aid only and no warranty of any kind is made as to completeness or correctness for your specific installation.

You can copy and modify these samples for your specific requirements. The libraries listed for searching must include the following in the order shown:

1. User libraries you may have defined
2. Product base libraries
3. CA Common Services for z/OS base libraries

Chapter 2: Using CA Hyper-Buf

This chapter provides information on the functions, constraints, utilities, and control statements used in the dynamic buffering process. CA Hyper-Buf dynamic buffering can be enabled or disabled at any time. You enable CA Hyper-Buf by running batch program GVBDBFON. You disable CA Hyper-Buf by running batch program GVBDBOFF.

There are sample job streams and procedures for these programs contained in the distribution CBS3JCL and CBS3PROC data sets. CA Hyper-Buf can also be enabled and disabled using the TSO interface. A full description of GVBDBFON and GVBDBOFF is included later in this manual.

This section contains the following topics:

CA Hyper-Buf Advantages

VSAM continues to be the most common file organization used in z/OS shops apart from relational databases such as IBM's DB2 and IMS, Advantage CA-Datacom, Advantage CA-IDMS and ADABAS. COBOL programs reading and writing VSAM files are a mainstay of your in-house written applications. These applications will continue to be used for many years to come and CA Hyper-Buf provides fully optimized access to VSAM data without manual tuning of VSAM parameters.

The buffering procedures and facilities used by VSAM to accelerate processing and provide maximum performance include the following:

- Dynamic VSAM buffering affects VSAM performance. Optimum buffering leads to improved performance, which results in shorter online response times, increased batch throughput, and shorter batch turnaround times. Improper buffering causes job backlogs, missed deadlines, and frustrated personnel.
- VSAM manages disk accesses on behalf of application programs. To do this, VSAM reserves blocks of virtual storage called buffers, which are used as staging areas for data that is on its way to or from disk.

Controlling Dynamic Buffering

CA Hyper-Buf allows you to place constraints upon the dynamic buffering process. Constraints are limits that you set. These limits are honored by the CA Hyper-Buf buffer calculation routines.

Through constraints you can include, exclude, and control the dynamic buffering process for specific SMS classes, jobs, programs, clusters, and program/ddname pairs. You can also place limits on the amount of virtual storage that CA Hyper-Buf is allowed to use for buffers, and you can establish acceptable ranges for most of the buffering variables.

You specify these constraints in a sequential file that is read by GVBDBFON when CA Hyper-Buf is initialized on your system. These controls are discussed in greater detail later in this manual.

You can also modify the constraints at any time by running the GVBREFR utility using either batch or the ISPF panels. This utility allows you to alter the constraints that apply without stopping and restarting CA Hyper-Buf.

Processing Considerations

There are times when you may want to benchmark performance with and without CA Hyper-Buf. You may have an application that uses large tables or arrays and you therefore wish to limit the amount of virtual storage that is dedicated to buffers. You may have applications that you specifically want to 'detune' so that they do not compete with other important jobs (possibly online or other time critical processing). Finally, you have yet other jobs that require the best possible performance.

For these reasons, you must be able to control how dynamic buffering functions in your environment. To provide control over all aspects of dynamic buffering, you can specify CA Hyper-Buf constraints at any of several levels. You can specify limits for particular clusters or set only overall system defaults. Alternatively, you can specify exactly the buffering to be performed for selected jobs, and let all non-selected clusters abide by the default values that you specify. These controls are not required. CA Hyper-Buf automatically optimizes buffers for you without requiring you to supply controls. The controls are provided so that you can modify the automatic buffering process because of the above or other reasons.

Your control over the dynamic buffering process is as general or as specific as you want it to be. Setting system default limits requires only a few control statements in the GVBDBFON control stream.

You can specify only the system defaults at first and let dynamic buffering work for you. Later, you can decide to select a few specific jobs, clusters, and programs that warrant individual attention and specify constraints that are customized specifically for them.

You can view the results of your constraints upon the buffering process by modeling the buffering process. This facility is provided by the ISPF dialogues that are supplied with this product.

Note: CA Hyper-Buf should not be run simultaneously with other VSAM or QSAM buffering products.

Processing Constraints

CA Hyper-Buf implements several levels of control at which you can specify constraints. These levels (from least specific to most specific) are as follows:

- SYSTEM defaults
- SMS CLASS NAMES
- CLUSTER NAME
- JOBNAME

- PROGRAM NAME
- PROGRAM/DDNAME pairs

Within each of the above levels, you can impose constraints on CA Hyper-Buf. Constraints include the maximum allowable amount of virtual storage that can be used at OPEN time for buffers, the minimum amount of virtual storage that you want left available in the region below 16M after the OPEN completes, the maximum number of data and index buffers that CA Hyper-Buf can select for an OPEN, and so forth, all of which is explained later in Chapter 3, "Constraint Control Statements."

Most of the constraints are available at each of the above levels. Any constraints that are restricted in any way are documented with the restrictions explicitly mentioned. This means that you can adjust the constraints for specific clusters, jobs, and programs if you wish, or you can set only the system level constraints that apply to all OPENs.

The system level constraints are used as the default for any OPEN constraints that are not specified at any of the other levels. If you do not supply a system default for any constraint, CA Hyper-Buf creates a default for you.

Fine Tuning

You may have specific applications that are extremely time critical or possibly applications that run in the ever shrinking night batch window. In these circumstances, you can increase the amount of virtual storage available for buffering to decrease the elapsed time for these jobs.

At other times, you may have important work that you do not want to interrupt, and you want to restrict the amount of buffering for any 'less important' jobs. Restricting buffering can detune these other jobs and reduce their CPU consumption, leaving more computing resources for the more important job(s). These detuned jobs consume approximately the same amount of CPU time, but over a longer elapsed time. These jobs become more I/O bound, which leaves more processor time for other jobs.

You can accomplish both tuning and detuning through the constraint control statements.

Dynamic Buffering Control Statements

You can exert as much or as little control as you wish over the buffering process. You exercise this control at initialization time through control statements supplied to GVBDBFON, the CA Hyper-Buf initialization program, and by refreshing the constraints using the GVBREFR utility after CA Hyper-Buf is initialized on your system. You can also exercise this same control using the TSO interface.

You control the dynamic buffering process by placing control statements in the GVBDBFON or GVBREFR SYSIN stream. These controls allow you to select both specific and generic instances of OPENs and to affect the number of buffers selected at OPEN time by dynamic buffering. You can specify exact numbers of buffers, set limits that CA Hyper-Buf will not exceed, or exempt entirely selected jobs clusters, and programs from the dynamic buffering process. The amount of control you exert at each and every level is up to you, and is optional.

There are several types of control statements available. These types are SYSTEM_DEFAULTS, MODE group controls, LEVEL selectors, and CONSTRAINT specifications. This section of the manual supplies an overview of the constraint capabilities, and the details of the SYSTEM_DEFAULTS, MODE control, and LEVEL selectors. The constraints are described in detail in the next chapter.

Processing Constraints

You control the CA Hyper-Buf dynamic buffering using CNTLFILE control statements supplied to GVBDBFON when you initialize the system, or to GVBREFR after CA Hyper-Buf is initialized on your system. The general format of the control file is as follows:

```
SYSTEM_DEFAULTS
  (list of  system  default constraints)
  . . .
  . . .
MODE=(   INCLUDE  | EXCLUDE  )
  CLUSTER=ABC
  (list of  cluster ABC  constraints)
  JOB=PAY+
  (list of  constraints for job  names beginning with  'PAY')
  (other  mode control group  constraints)
  . . .
  . . .
MODE=(   EXCLUDE  | INCLUDE  )
  (other  mode control group  constraints)
  . . .
```

Indentation of the various control statements is optional. Controls are indented in these examples to facilitate visual perception of the hierarchy of the rules. The indentation does not affect the hierarchy. The hierarchy is fixed by CA Hyper-Buf.

There are two modes available, INCLUDE and EXCLUDE. EXCLUDE mode specifies levels which are exempt from CA Hyper-Buf control and follows normal VSAM buffer selection. INCLUDE specifies levels eligible for buffer modification by CA Hyper-Buf.

Within these two modes, you can specify LEVEL selectors. These allowable level selectors are shown below:

- CLUSTER=name
- SMSclass=name
- JOB=name
- PROGRAM=name
- DDNAME=name

The name fields represent constraint variables that are matched to the corresponding OPEN variables for each VSAM OPEN. For example, the JOBNAME of the job that is issuing the OPEN is compared to all of the JOB= level selectors by the CA Hyper-Buf OPEN intercept routine. Any constraints that are specified at a matching level affect the CA Hyper-Buf buffer calculations.

Within each of the levels you can specify buffering CONSTRAINTS. Constraints include maximum allowable BUFFERSPACE, BUFNI, and BUFND and corresponding minimum allowable values for BUFFERSPACE, BUFNI, and BUFND, as well as other tunable values. The constraints associated with each matched level selector are merged into a composite constraint table by the CA Hyper-Buf OPEN intercept routine.

All controls can be specified in free form. Only positions 1-72 of each control record are scanned for control information. Numeric fields, when required, can be suffixed by the letter K to indicate multiplication by 1024. For example, specifying 60K is equivalent to specifying 61440.

You must specify command/operand pairs on the same record. You cannot specify MAXBUFSP on one record, followed by 60K on the next record, and expect MAXBUFSP to be set to 60K. Both fields of the control must be on the same logical record of the control file. You can specify as many records as required to contain all of your controls.

Method of Operation

Certain types of OPENs cannot be buffered by CA Hyper-Buf. The excluded types of OPENs are:

- LSR/GSR OPENs. These are excluded because the buffers for these types of files are allocated before the ACB is opened. CA Hyper-Buf cannot affect the number of the pre-allocated buffers.

  Note: CA Hyper-Buf can convert NSR requests to LSR, if requested, in which case CA Hyper-Buf does control LSR buffering).

- UBF. These are ACBs that specify user buffering. Since the application controls these buffers, CA Hyper-Buf has no control over the number of buffers allocated.

- Improved Control Interval Access (ICI).
- Control Blocks in Common (CBIC).
- Innovation's IAM type files are bypassed.

After you initialize CA Hyper-Buf, all VSAM OPENs that are not excluded by the above restrictions are screened by the CA Hyper-Buf OPEN intercept routine. This intercept routine compares the cluster name, SMS class names, job name, program name, and program/dd name combinations against the LEVEL selectors that you supplied to CA Hyper-Buf at initialization time. Any constraints attached to any matching level selector are merged into a composite constraint table.

The composite constraint table is built by the OPEN intercept routine. This table is a composite of all of the constraints associated with all matching LEVEL selectors. As each of the LEVELs is checked by the OPEN intercept routine, the constraints associated with each matching level are merged into the composite constraint table.

If no matching LEVEL selectors are found (you did not code any matching CLUSTER, SMSclass, JOB, PROGRAM, or PROGRAM/DDNAME LEVEL selectors), or any constraint not specified in any matching LEVEL selectors (for example, only JOB matched, but you did not specify MAXBUFSP at the JOB level), the corresponding SYSTEM DEFAULT values are used in the composite constraint table.

Note that LEVEL SELECTORS are independent entities. Independent entities mean that a particular level selector is never owned or controlled by any other level selector. The only exception to this rule is the DDNAME selector which is owned by the PROGRAM level selector that precedes it.

After CA Hyper-Buf checks all levels and merges all of the constraints specified at each matching level, CA Hyper-Buf uses the resulting composite constraint table, along with the CI size(s), number of index levels and index set records, and OPEN options (random, sequential, mixed), to determine the optimum number of index and data buffers for this particular OPEN.

Establishing the MODE

There are two modes available, INCLUDE mode and EXCLUDE mode. You can specify the MODE by including one or more MODE control statements in the GVBDBFON or GVBREFR SYSIN stream. Also, there is a SYSTEM default mode available through the DEFAULTMODE system constraint.

Once you specify a particular mode, that mode remains in effect until you specify another mode card in the control stream (see the sample GVBDBFON and GVBREFR SYSIN streams in Chapter 5, "Batch Utility Programs").

The MODE is specified simply as MODE=INCLUDE or MODE=EXCLUDE. Each LEVEL selector belongs to the MODE control statement that precedes it, or the system default mode if there is no MODE established prior to the level selector. All LEVEL selectors that follow a MODE control up to the next MODE control (or the end of file) are called a mode control group. The mode control group is an INCLUDE mode control group, if the owning mode card specifies INCLUDE; otherwise the group is an EXCLUDE mode control group.

At VSAM OPEN time, the intercept routine compares the cluster name, SMS class name(s), job name, program name, and the program/dd name combinations of the OPEN issuer against a table that was built from your control statements at initialization time, or modified at refresh time. If a matching LEVEL SELECTOR is found in any EXCLUDE mode control group, this OPEN is not dynamically buffered. The OPEN intercept routine passes control directly to normal VSAM OPEN. Any constraints associated with an excluded level selector are ignored.

Excluded OPENs

An explicitly coded matching EXCLUDE is always honored. For example, if the job name and cluster name are both in an INCLUDE mode control group, but the program name is in an EXCLUDE mode control group, this OPEN is NOT dynamically buffered. Any explicit matching EXCLUDE always takes precedence over all other constraints.

If you specify MODE=EXCLUDE then list several CLUSTER level selectors, all of the clusters named are exempt from dynamic buffering. Normal VSAM buffering is in effect for those clusters and not dynamic buffering. This is true regardless of which job or program might access these clusters.

If there are no matching LEVEL statements found in any INCLUDE or EXCLUDE mode control group at OPEN time, the SYSTEM default mode prevails. The SYSTEM mode is the default mode and is specified by the DEFAULTMODE control statement in the SYSTEM constraints section of the initialization control stream.

Assume that DEFAULTMODE=EXCLUDE is specified in the system defaults and you have several CLUSTER level selectors specified after a MODE=INCLUDE control, but none of the CLUSTERs that were specified matches the cluster currently being opened. This cluster is exempt from dynamic buffering (assuming no matches are found on job or program names). Because no match was found in the INCLUDE mode control groups, the default mode prevails.

The default mode in this case is EXCLUDE, so dynamic buffering is not in effect for this OPEN. The DEFAULTMODE=EXCLUDE does not override explicitly coded LEVEL controls in an include mode control group. Default mode is used only when no matching level selectors are found.

ACCBIAS, a subparameter of the AMP parameter, is used to specify the type of System Managed Buffering used by a file. Because System Managed Buffering is a facility of IBM's System Managed Storage CA Hyper-Buf will not provide dynamic buffering when this parameter is specified. However, you will receive message GVB676I denoting this condition.

If the data set being opened is determined to be controlled by Innovation Data Processing's IAM, message GVB677I is issued and the data set is not processed by CA Hyper-Buf.

Level Selectors

Within each mode control group, you can specify as many level selectors as you wish. Level selectors 'belong' to the previous MODE control statement or the DEFAULTMODE control statement if no previous MODE statements exist in the input stream. All of the level controls that belong to a particular MODE card are called a mode control group.

The level controls that you can specify are SMS class, CLUSTER, JOB, PROGRAM, and PROGRAM/DDNAME pairs. Any OPEN that matches any LEVEL selector(s) included within an EXCLUDE mode control group is exempt from dynamic buffering.

At OPEN time, the intercept routine initializes a constraint table with the system default values. Then each of the control levels is searched for any matches (job level searched using current job name, and so forth). The constraints associated with each matching control level entry are merged into the constraint table, which overlays any data that may have already existed for that particular constraint (like maximum buffer space).

After all of the levels have been searched, the resulting constraint table, called the composite constraint table, is passed to the dynamic buffering calculation routines. This composite table governs the calculation of buffers for this OPEN.

Level Descriptions

Each of the levels is checked in the following order:

- SYSTEM_DEFAULTS. The lowest priority specification. Any constraint(s) specified on any matching level selector will override corresponding values specified here. The values specified here are used only if none of the matching level selectors specify a corresponding constraint. For example, the SYSTEM_DEFAULT specified value for MAXBUFSP is used only if no matching level selector specifies MAXBUFSP.
- SMS classes (SCLASS, MCLASS, DCLASS), from low to high precedence.

  This is the least specific level that you can specify (except for the system defaults). Many clusters may be defined in each class. Each cluster defined in a particular class can be accessed by several jobs, each of which is composed of several job steps. Each of these job step programs performs a different type of processing against the cluster. Therefore, limits specified at the SMS class level are not specific, but general guidelines for dynamic buffering to observe, unless overridden by a more specific level selector match.
- CLUSTER. This level is only slightly more specific than SMS storage, management, or data class. Many jobs can access this cluster. Each job can have several steps or programs that may access this cluster. All of these programs and jobs perform different types of processing against this cluster. Therefore, limits specified at the cluster level are only slightly more specific than SMS classes.
- JOBNAME. This is more specific than cluster. A job can be composed of several steps, and some or all of these steps can process different clusters. Each program is probably related to the others in that they are all probably part of the same application system (such as, payroll) and may or may not perform similar processing. For example, step one of a job might randomly update a master file. Step two of the same job might sequentially copy that same master file to tape.
- PROGRAM name. More specific than job name. A particular program may access more than one VSAM file, with different types of processing for each file. For example, one of the clusters may be a master file which is processed randomly, and another cluster a journal file that is processed sequentially.

  Both the program name on the EXEC card and the program name on the CDE are checked.

- DDNAME combined with program name. This is the most specific level available. A particular program (for instance, the PAYROLL program) always performs the same type of processing against the cluster that is allocated by a particular DD card (such as, the PAYMSTR DD). This is true regardless of which cluster is named on that DD card, or which job is running this program. For example, a PAYROLL program may be used by a service bureau to create paychecks for many customers. Each customer has a separate master file, so this program is run many times, each time with a different cluster specified on the PAYMSTR DD card.

  However, these clusters are all similar in that each has the same record layout (if not, the payroll program would certainly encounter problems), record length(s), and key specification, and so forth. The processing is approximately the same. The payroll program might read sequentially through the file and update year-to-date values in the records as it creates the paychecks. This is true for any PAYMSTR DD cluster.

Specifying LEVEL Selectors

A level selector controls the scope of the constraint card(s) which follow it. Also, the LEVEL selectors determine the precedence of the rules (for example, which constraints override which other constraints).

This section discusses the available LEVEL selectors and explains their use.

SMSCLASS Level Selectors

The level selector allows you to associate constraints with specific SMS classes. Any clusters that are defined with specified classes acquire the constraints associated with the matching classes.

The SMSCLASS level is actually composed of three individual sub-levels. The three SMS level selectors are (from low to high precedence):

- SCLASS for SMS storage class
- MCLASS for SMS management class
- DCLASS for SMS data class

You specify the SMS level selectors as:

```
DCLASS=DOGS
(constraints for any  cluster  defined in the   DOGS  dataclass)

MCLASS=BARK
(constraints for any  cluster  defined in the   BARK  management class)

DCLASS=BITE
(constraints for any  cluster  defined in the   BITE data class)
```

Note that a cluster can belong to none, one, two, or all three SMS classes. All matching SMS class constraints are merged for each cluster.

You can specify constraints for as many SMS classes as you may have defined in your shop. You can even define SMS class constraints for SMS classes that you have not yet defined on your system, but plan to.

If you code SMS constraints, and SMS is not active on your system when you activate CA Hyper-Buf, you get a warning message to alert you to this fact. The warning does not affect CA Hyper-BufCA Hyper-Buf safely ignored - although you may want to start the SMS subsystem, if that is your intention.

The SMS class names that you specify can be generic or full class name specifiers. Generic names are discussed more fully on page 2-15 for Clusters.

DCLASS: Associating Constraints with SMS Data Class

Restriction: Valid only on z/OS systems with SMS installed and active. This option generates a warning if specified in a non-SMS environment.

You can assign constraints to an SMS data class. Any constraints that you specify are implemented for any cluster that is defined with that data class.

```
DCLASS=SMSdataclassname
 ...(any  constraints you  wish)
```

In this example, *SMSdataclassname* is the name of an existing or future data class. All DCLASS constraints should be defined within an INCLUDE mode control group, unless you want the DCLASS to force exclusion.

See related topics MCLASS and SCLASS in this section.

MCLASS: Associating Constraints with SMS Management Class

Restriction: Valid only on z/OS systems with SMS installed and active. This option generates a warning if specified in a non-SMS environment.

You can assign constraints to an SMS management class. Any constraints that you specify are implemented for any cluster that is defined with that class.

```
MCLASS=SMSmgmtclassname{
 ...(any  constraints you  wish)
```

In this example, *SMSmgmtclassname* is the name of an existing or future management class. All MCLASS constraints should be defined within an INCLUDE mode control group, unless you want the DCLASS to force exclusion.

See related topics DCLASS and SCLASS in this section.

SCLASS: Associating Constraints with SMS Storage Class

Restriction: Valid only on z/OS systems with SMS installed and active. This option generates a warning if specified in a non-SMS environment.

You can assign constraints to an SMS storage class. Any constraints that you specify are implemented for any cluster that is defined with that class.

```
SCLASS=SMSstorclassname
 ...(any  constraints you  wish)
```

In this example, *SMSstorclassname* is the name of an existing or future storage class. All SCLASS constraints should be defined within an INCLUDE mode control group, unless you want the SCLASS to force exclusion.

See related topics DCLASS and MCLASS in this section.

CLUSTER LEVEL Selector

The CLUSTER level selector allows you to target specific or generic clusters. You specify the CLUSTER level selector as:

```
CLUSTER=PAYROLL.MASTER+
(constraints for any    'PAYROLL.MASTER'  file)
```

The cluster name can be a full cluster name (all characters) or a generic cluster name.

Generic names specify the first fragment of the name, followed by a plus sign to indicate a variable number of "don't care" characters (but at least one character). For example CLUSTER=PAY+ matches all four character or longer cluster names that start with the first three characters PAY. This specification includes clusters with names like PAYROLL.MASTER.FILE, PAY.AUDIT.TRAIL, and PAY.HISTORY.FILE as well as others.

Note: The fragment specified for a generic name can be any portion of the name. For example, you do not have to 'break' a cluster name fragment at the end of a qualifier.

JOB LEVEL Selector

The JOB level selector allows you to select specific JOBs, either by specifying a full JOB name or a generic job name ending with a plus sign. See the CLUSTER LEVEL Selector section for a description of generic names. The following is an example:

```
JOB=P1225+
(constraints for any  6  character
or   longer job  name that starts with  'P1225')
```

PROGRAM LEVEL Selector

The PROGRAM level selector allows you to target specific or generic programs. Generic program names are specified the same way as generic clusters. The following example selects program A225772, no matter which JOB is executing that program, or which clusters the program accesses.

```
PROGRAM=A225772

(constraints for program  A225772)
```

Both the program name on the EXEC JCL card and the program name on the current CDE are matched against the PROGRAM constraints. CA Hyper-Buf can then apply constraints to programs that are invoked by the job step (EXEC card) and programs that are invoked later using LINK, ATTACH, or XCTL. You can invoke different constraints for a SORT that is LINKed. This allows SORT to have different constraints from the rest of the job step.

The matching EXEC constraints are merged with the CDE constraints by CA Hyper-Buf. Any constraints that are specified on the matching CDE program name overrides the corresponding constraints for the EXEC program, if any.

This feature does not allow CA Hyper-Buf to detect a program that has been loaded, then called. LOADed programs leave no trace of when they are entered or exited, so CA Hyper-Buf cannot determine which routine is actually in control at the time of the OPEN or CLOSE. In this case, the LOADed/CALLed program has exactly the same constraints as the caller.

PROGRAM/DDNAME Combined Selector

Following a PROGRAM control statement, you can specify one or more DDNAME selectors that are associated with that program. For example, you can list some or all of the DDNAMEs (for VSAM files) that this program uses. If you assume that program A225772 requires a VSAM input file on the MASTER DD card and a VSAM output file on the OUTPUT DD CARD, you could specify these DD statements following the PROGRAM selector.

```
PROGRAM=A225772

   (overall  constraints for program  A225772)

   DDNAME=MASTER

   (specific constraints for file  attached to MASTER   DD  card)

   DDNAME=OUTPUT

   (specific constraints for file  attached to OUTPUT   DD  card)

   DDNAME=INPUT

   (specific constraints for file  attached to INPUT   DD  card)
```

Note: These ddname constraints apply only to program A225772.

You do not have to specify all of the DD statements that the program uses. You can specify only those DD statements for which you desire values other than the composite values assigned.

Because of the constraint merge, any constraints specified following a DDNAME level selector override any duplicate constraints at the program, job, and cluster levels.

Chapter 3: Constraint Control Statements

Constraint control statements allow you to control the buffering process. You can specify very restrictive rules, which limit the effects of dynamic buffering, or you can allow dynamic buffering a free hand in determining the number of buffers to allocate. You can be conservative with some level selectors and liberal with others. The amount of control exercised is entirely up to you. You can even choose not to code any constraints and allow the SYSTEM_DEFAULTS to prevail for every VSAM OPEN.

This section contains the following topics:

Specifying Constraint Control Statements

The constraints are described in the following section. Almost all of these constraints can be specified following any LEVEL selector and will apply to that specific LEVEL selector. Any constraints that are restricted to particular LEVELs are documented as such.

Sample Control Stream

You can think of constraints as being owned by or attached to the preceding level selector. This means that you can specify MAXBUFSP (or any other constraint) for JOBs, CLUSTERs, PROGRAMs, or PROGRAM/DD combinations. See the following control stream example:

```
SYSTEM_DEFAULTS*  defaults
   MINBUFNI=5
   MAXBUFND=20
   MINBUFND=5  MAXBUFSP=81920
   MINBUFSP=12K
   DEFAULTMODE=EXCLUDE
   STATS
   . . . .
   . . . .

MODE=INCLUDE    *  Begin  INCLUDE    MODE CONTROL  GROUP
   CLUSTER=PAY+*  CLUSTER  level selector constraints
      MAXBUFSP=70K    MAXBUFND=28
      MINBUFNI=8
   CLUSTER=PAYROLL.MASTER.FILE  *  CLUSTER  level and  constraints
      MAXBUFSP=40960    MAXBUFND=8
      MINBUFNI=3
      MAXBUFNI=6
   PROGRAM=F28945       *  PROGRAM  level and  constraints
      MAXBUFSP=32768
      DDNAME=MSTRIN     *  DDNAME  and  constraints
         MAXBUFSP=60K
   . . . .
```

Constraint Merging

The sample control stream in the previous section shows that cluster PAYROLL.MASTER.FILE and program F28945 will both be dynamically buffered because they are contained within an INCLUDE mode control group (assuming that no other EXCLUDE level matches, which takes precedence). When cluster PAYROLL.MASTER.FILE is OPENed, CA Hyper-Buf looks at the OPEN options (random, sequential, and so forth) and cluster information (index set records, CI sizes, device type, and so forth). Then CA Hyper-Buf determines an optimum number of buffers that does not exceed a total of 40960 bytes of virtual storage, has no more than eight data buffers, and has at least three index buffers.

These numbers are determined by simulating the actual checking done by CA Hyper-Buf.

1. The composite constraint table is initialized with the system defaults: MAXBUFND=20, MAXBUFSP=81920, and so forth, including all of the system default values that are specified.
2. The CLUSTER selectors are compared against the cluster currently being opened. Cluster PAYROLL.MASTER.FILE matches a cluster specification exactly, and the corresponding constraints are merged into the composite table. The composite table now contains MAXBUFND=8 (overlaid) and MAXBUFSP=40960 (overlaid), in addition to the original default values.

   Notice that the cluster does **not** match on the generic PAY+ level selector. At each level, CA Hyper-Buf chooses the selector that best matches the parameter.
3. CA Hyper-Buf compares the job selectors, and any matching entry constraints are merged into the composite table.
4. CA Hyper-Buf compares the program level selectors, and any matching entry constraints are merged into the composite table. Finally, program/ddname selectors are examined for matches, and the constraints are merged.

In this example, the maximum allowable number of index buffers is not specified on any matching level selector; only the minimum allowable. Since there is no specification made for the maximum index buffers, the system default value of STRNO + (number of index set records) is put into the composite table. Moreover, the cluster specifies no other minimum values, so all of the system default minimums are still in effect.

When any VSAM file is opened, each of the instance parameters (SMS class, cluster name, job name, program name, program/ddname) is compared against the corresponding LEVEL selectors. If a match is found, the constraints associated with the matching LEVEL selector are merged into a composite constraint table for this OPEN.

This merging of constraints is necessary because JOB F28945 may open cluster PAYROLL.MASTER.FILE, and both of these LEVEL selectors matches, and each level has different constraints. The merge procedure sets MAXBUFSP=32768 (overlaid), MAXBUFND=8, and MAXBUFNI=6 for this OPEN.

If a matching DD entry is encountered, the composite constraint table is again modified. If the cluster PAYROLL.MASTER.FILE is on the MSTRIN DD card, MAXBUFSP is overlaid with 60K. The resulting composite constraint table is: all minimum values are the same as system defaults (except MINBUFNI, which is 3) and the maximums are MAXBUFSP=61440 and MAXBUFND=8.

After all LEVEL selectors are searched and matching LEVEL constraints are merged, the resulting composite constraint table governs the selection of buffers.

Constraint Specifications

Most of the constraints allow you to specify a maximum value and a corresponding minimum value. For example, you can specify both a maximum and a minimum allowable buffer space. CA Hyper-Buf implements a value that falls between the specified high and low values, as long as it does not conflict with other constraints.

If you specify MINBUFND=10, MAXBUFND=50, and MAXBUFSP=60K, CA Hyper-Buf uses between 10 and 50 data buffers, as long as the resultant buffer space does not exceed 60K.

Note: The index component also uses part of this buffer space, so you may not actually have 60K of data buffers).

Any user specified MINxxx constraints are validated by the CA Hyper-Buf OPEN intercept routine. The user specified MINxxx constraints are compared with the minimum VSAM allowable values. If the user specified values are less than the VSAM allowed minimum values, the user values are increased to the VSAM minimum.

For example, if an ACB specifies STRNO=4 and is associated with a cluster that has a 4K data CI size and a 2K index CI size, the true minimum allowable VSAM buffer space (NSR) for this file is 28K. If the user has specified MINBUFSP=12K, the CA Hyper-Buf OPEN intercept routine overrides the user value, and use 22K as the MINBUFSP value.

If any minimum constraint value exceeds the corresponding maximum constraint value (after the constraint merge is complete), the maximum constraint is increased to the value of the minimum constraint. For example, if after all constraint merging is complete, MAXBUFSP=40K and MINBUFSP=60K, then MAXBUFSP is increased to 60K, which is the MINBUFSP value.

In most cases, you do not need to specify any minimum values. CA Hyper-Buf calculates an optimum number of buffers, without exceeding any maximum value that you have specified. For example, assume that the composite constraints are MAXBUFSP=52K, MAXBUFND=8, data CI size is 10K, and no index buffers are required (for example, an ESDS file). CA Hyper-Buf can assign five data buffers for a total of 50K buffer space (depending upon devicetype, OPEN options, and so forth), which is as close to MAXBUFSP as possible with this particular file. Specifying a minimum value has no effect in this case.

Consider setting minimums if you specify SEQUENTIAL, RANDOM, or CICS constraints for any level. For example, if you specify the SEQUENTIAL processing constraint for a KSDS because access is predominantly sequential, CA Hyper-Buf acquires only one index buffer per string and as many data buffers as reasonable. This processing is optimized for sequential processing. If the application also has some amount of random processing, you may be able to improve performance by increasing the number of index buffers. You can ensure that more index buffers are allocated by specifying a larger MINBUFNI.

Each constraint is associated with the LEVEL SELECTOR that precedes it in the control stream.

Constraint Descriptions

A description of each of the constraints follows.

Defaults shown are values that are used if you have not specified anything for this constraint. These values are internal CA Hyper-Buf defaults that are used if you do not specify any other values.

ACBOVR: Overriding ACB Specified Buffer Values

Default: No ACBOVR

The ACBOVR constraint specifies whether CA Hyper-Buf overrides ACB buffer values. Specifying ACBOVR causes CA Hyper-Buf to ignore ACB specified values for BUFNI, BUFND, and BUFSP.

If ACBOVR is not specified, ACB values are honored by CA Hyper-Buf, and all calculated CA Hyper-Buf values are discarded.

AMPOVR: Overriding AMP Specified Buffer Values

Default: AMPOVR=NO

The AMPOVR constraint specifies whether CA Hyper-Buf is to override user coded AMP parameters that are supplied on the DD card. Specify AMPOVR=key to control the way that CA Hyper-Buf treats user specified AMP values.

Allowable values for *key* are:

- FORCE
- GREATER

- NO
- MAX

AMPOVR=FORCE causes CA Hyper-Buf to always override AMP specified values. Any values specified in the AMP parameter on the associated DD card are ignored, and the OPEN proceeds with the CA Hyper-Buf calculated values for BUFNI, BUFND, and BUFFERSPACE.

AMPOVR=GREATER causes CA Hyper-Buf to override AMP specified values only if the CA Hyper-Buf calculated buffer space exceeds the AMP specified buffer space (or the buffer space calculated using the AMP values, if buffer space is not specified explicitly in the AMP parameter). If the CA Hyper-Buf buffer space is greater, OPEN continues with the CA Hyper-Buf values for BUFNI, BUFND, and BUFFERSPACE, and the AMP values are discarded. If the AMP buffer space is greater, the AMP values are used for this OPEN, and the CA Hyper-Buf values are discarded.

AMPOVR=NO (or if AMPOVR is not specified at all) causes all CA Hyper-Buf values to be discarded if any AMP values are specified on the DD card.

AMPOVR=MAX causes the OPEN intercept routine to compare the CA Hyper-Buf calculated values of BUFNI, BUFND, and BUFFERSPACE against the corresponding AMP values. In each case, the larger value is selected, and used for this OPEN. The MAXBUFSP parameter is ignored if MAX is coded for AMPOVR. The larger values of the BUFNI and BUFND parameters are used regardless of the resulting buffer space.

Note: MAX differs from the GREATER option, because the GREATER option uses either the AMP values or the CA Hyper-Buf calculated values, depending upon the comparison of the two buffer space values. This could result in a reduction of BUFNI, BUFND, or both.

For example, the user can specify BUFND=10 in the AMP parameter on the DD card of a file that is processed randomly. If the cluster is a KSDS, 4K data and 4K index, and specifies STRNO=1, the implied AMP buffer space is 45,056 (10 x 4K + 1 x 4K). The CA Hyper-Buf calculated values can be BUFND=2, BUFNI=18, BUFSP=81920.

If AMPOVR=GREATER is specified, the OPEN intercept uses the CA Hyper-Buf calculated values, since the CA Hyper-Buf buffer space is greater than the AMP buffer space. This causes a decrease in the BUFND value (from 10 to 2) and an increase in the BUFNI value (from 1 to 18) compared with the AMP specified values. The extra index and fewer data buffers is consistent with random processing, and performance should be improved, even though the data buffers are decreased.

In this same situation, AMPOVR=MAX results in BUFND=10, BUFNI=18, BUFSP=114,688. The larger values of BUFNI and BUFND are retained, and the new buffer space is calculated. The user specified MAXBUFSP is ignored in this calculation.

If AMP is not coded on a DD card, CA Hyper-Buf calculates buffers for INCLUDE mode OPENs.

ACCBIAS, a subparameter of the AMP parameter, is used to specify the type of System Managed Buffering used by a file. Because System Managed Buffering is a facility of IBM's System Managed Storage CA Hyper-Buf will not provide dynamic buffering when this parameter is specified. However, you will receive message GVB676I denoting this condition.

CICS: Restricting Buffering for the CICS Environment

Restriction: Must follow a PROGRAM level selector.

Default: No CICS

The CICS constraint may be applied to any PROGRAM level selector. This constraint alters the algorithm which determines how many buffers to use for a file which is opened for mixed mode (random and sequential) processing. Without this constraint specified, both the random and sequential options would be weighed equally when determining the number of buffers to assign. With this constraint, the random option is assigned a higher weight than the sequential option.

This makes better use of virtual storage in an online system environment (like CICS) when both random and sequential requests are issued for a file, but the random requests are much more frequent than the sequential requests.

This constraint has no effect on any files that are using shared resources.

CUSHION: Reserving Storage Below 16M

Default: CUSHION=100K

CUSHION allows you to guarantee that some virtual storage is left below the 16M line in your region for non-buffering purposes. Dynamic buffering tries to guarantee that at least CUSHION bytes remain in the region after the buffers are allocated for OPEN. Specify as:

```
CUSHION=value
```

If any LEVEL selector does not specify the CUSHION constraint, the SYSTEM default CUSHION value is used.

If you do not supply a SYSTEM default CUSHION, 100K is used as the cushion.

See the related constraint DYNAMIC: Controlling Dynamic Region Sizing (see page 46).

DEFER: Deferring WRITEs

Restriction: DEFER requires LSR to be effective. You can specify DEFER even if you do not specify LSR, and no error is generated. CA Hyper-Buf may convert strictly random access OPENs to use LSR buffers, and GVBDBFON cannot tell at initialization time how any particular cluster is accessed. If you specify DEFER when LSR is not in effect, normal VSAM buffer writes occur, and DEFER is ignored.

Default: No DEFER

Normally, VSAM writes buffered CIs to disk immediately when any data in the CI is altered. This is true for alterations caused by record insertions and record updates, including deletes.

Applications that insert or alter many logical records that reside in the same CI may avoid multiple WRITEs of the same CI by specifying DEFER.

DEFER causes VSAM to update the CI in the buffer and not write it to the disk until either the application forces a write (CLOSE, WRTBFR, and so forth) or until the buffer is required for a new CI to be read into storage.

This option can drastically reduce the number of I/O operations required to update a file randomly if the updates are clustered around specific keys in the file. This reduction in I/O can significantly improve the elapsed times for jobs that make multiple updates to the same CI.

Implications to Consider Before Using DEFER

Since it can be a long time (relative to CPU speeds) before the buffer is actually written to disk, it is possible that you will lose some data if the application abends or the processor dies (from a power failure, IPL, and so forth). In these situations, there may be data that has been updated in the buffer and not yet written to disk at the time of the failure. These updates may be lost.

If your processor never fails and your programs (including the operating system and its components) never abend, then you have no exposure. The less stable your system is, the greater your exposure.

For these reasons, you can exclude critical files (like journal and log files) from DEFER processing. If you do not request DEFER, normal write-as-you-go logic is used.

DEFINE: Creating a Named Constraint Prototype

The DEFINE keyword allows you to assign a name to a group of constraints. This group of constraints can be invoked multiple times following the definition of the prototype.

The format of the DEFINE command is:

```
DEFINE=name

   (any   constraints you  wish)
```

The field name is the name that you wish to assign to the group of constraints. This name can be any alphanumeric characters. The maximum length allowed for a name is 44 characters.

You specify the USE keyword to invoke a prototype. See related topic USE: Invoking a Constraint Prototype (see page 65).

The constraints specified for a prototype definition can include the USE keyword. The name specified on any USE statements must have been previously defined. DEFINEs can be nested 16 deep in this way.

This example assigns any constraints that you specify to the name name. These constraints may be referred to in later SMS class, JOB, PROGRAM, DDNAME, or CLUSTER levels.

Note: If the DEFINE occurs in an EXCLUDE mode control group, invoking this prototype EXCLUDEs the current level. Therefore, you should always define a prototype inside an INCLUDE mode control group, unless your intention is to create a synonym for EXCLUDE.

DSNSHARE: Share VSAM Control Block Structure

Default: DSNSHARE=YES

Option: DSNSHARE=NO

Specify YES to share VSAM control block structure. YES causes different DDs pointing to the same cluster to use the same VSAM control block structure, including buffers. This can reduce I/Os and allow updates performed by one DD to be available immediately by the other.

DSNSHARE=NO does not share VSAM control block structure.

DYNAMIC: Controlling Dynamic Region Sizing

Default: DYNAMIC=YES

Option: DYNAMIC=NO

Use the DYNAMIC keyword to control CA Hyper-Buf region size adjustment capabilities. If you specify DYNAMIC=YES, CA Hyper-Buf adjusts the region size by the amount of buffer space that CA Hyper-Buf allocates. This avoids X78 and X0A abends associated with region sizes that are too small, and avoids clusters that are not buffered because of the CUSHION. This option allows for larger buffer spaces, without forcing you to alter existing REGION parameters on your jobs.

DYNAMIC=NO disables dynamic region sizing.

EXCLDD: Globally Exclude a DDNAME

Restriction: The EXCLDD constraint may only be specified in SYSTEM_DEFAULTS.

Use the EXCLDD constraint to direct CA Hyper-Buf to bypass processing desired DDNAMEs. Up to 150 EXCLDD statements can be specified.

The EXCLDD keyword allows you to globally exclude a DDNAME.

The format of the EXCLDD command is:

```
EXCLDD=DDNAME
```

The DDNAME can also be specified as DDNAME+ which would match any DDNAME starting with 'DDNAME'.

FIX: Page Fixing LSR Buffers and IOBs

Restriction: The FIX constraint is effective only if specified for an OPEN that uses LSR buffering. Since CA Hyper-Buf attempts to use LSR processing for random access OPENs as a default, this parameter cannot be validated for applicability by GVBDBFON or the ACTIVATE function (ISPF). No error message is generated, but if FIX is specified for a cluster that does not use LSR buffering, FIX is ignored.

Default: UNFIX

The FIX constraint allows you to page fix VSAM LSR buffers and IOBs. This specification eliminates page faults during VSAM processing and may improve performance.

This option may also degrade performance in a severely storage constrained environment. If your system is already thrashing, do not use the FIX option, as this commits even more real storage.

FIX is the default for buffers that are backed in expanded storage (HIPERSPACE constraint specified) if the address space is also non-swappable (SWAP not specified). See the HIPERSPACE constraint description in the section HIPERSPACE: Backing LSR Buffers in HIPERSPACE (see page 47) for further information.

There are no operands for FIX. You can find an example of the FIX constraint in the section UNFIX: Keeping LSR Buffers and IOBs Pageable (see page 64).

See the section UNFIX: Keeping LSR Buffers and IOBs Pageable (see page 64) for more information about the UNFIX parameter.

HIPERSPACE: Backing LSR Buffers in HIPERSPACE

Restriction: Only buffers that are a multiple of 4K are eligible for backing in expanded storage. If the LSR pool contains some buffers that are a multiple of 4K and some that are not, only the buffers that are a multiple of 4K are backed in expanded storage. If none of the buffers are a multiple of 4K, no buffers are backed in expanded storage.

IPERSPACE is only effective if specified for a file that is using LSR buffering. Files that are strictly RANDOM use LSR processing, as will files that specify the LSR or SHRPOOL constraints. Since GVBDBFON cannot determine how a file is used, it cannot validate the use of HIPERSPACE. Therefore, you can specify HIPERSPACE anywhere (for example, no error message generated). However, the specification is effective only if LSR is in effect.

Default: HIPERSPACE=1

Option: HIPERSPACE=0 (turns off feature)

The HIPERSPACE control specifies that the LSR pool associated with this OPEN is to be backed in expanded storage. Access to expanded storage is slower than access to main memory, but much faster than access to DASD. By backing LSR buffers in HIPERSPACE, large quantities of data can be accessed without I/O.

The HIPERSPACE control specifies a multiplier that is used to determine the number of buffers that are allocated in expanded storage. CA Hyper-Buf attempts to allocate (multiplier x LSR buffers) HIPERSPACE buffers for this OPEN. The multiplier ranges from 1 to 32,767.

```
. . .

CLUSTER=A.Z,
   SHRPOOL=5,  *  Could  also be  LSR, or  merged  at a  lower  level
   HIPERSPACE=10
. . .
```

These constraints associate cluster A.Z with LSR pool 5. The LSR buffers are backed in expanded storage (HIPERSPACE). The number of buffers that are allocated in expanded storage are ten times the number of buffers that are allocated in the LSR pool.

This calculation is performed for each size buffer that is allocated to the LSR pool. For example, if the LSR pool 5 contains twelve 4K buffers, five 8K buffers, and two 16K buffers, and HIPERSPACE=10 is specified, then there are 10*12=120 4K buffers, 10*5=50 8K buffers, and 10*2=20 16K buffers allocated in expanded storage for this LSR pool.

HIPERSPACE backing of buffers in expanded storage creates a cache of data that can be accessed without an I/O to DASD. This can significantly reduce the number of I/Os required to process a cluster. If the requested data is resident in a HIPERSPACE buffer, it is moved synchronously to the requester's address space instead of being read asynchronously from DASD.

The data that is cached in HIPERSPACE buffers is not paged out. If the address space is swapped out, the data in the HIPERSPACE buffers must be rebuilt when the address space is swapped back in. The cache is lost each time the address space is swapped out. Therefore, swap-outs defeat the intended purpose of the cache, which is to reduce I/O. You should allow address spaces using HIPERSPACE buffers to be non-swappable for maximum performance.

CA Hyper-Buf automatically makes any address space using HIPERSPACE non-swappable, and automatically page fixes the LSR buffer pool. This is recommended for the above performance reasons. If for any reason you want to use HIPERSPACE and wish the address space to remain swappable, specify the SWAP constraint in addition to the HIPERSPACE constraint. This is not recommended for performance reasons, so be careful when you use this option in conjunction with the HIPERSPACE option.

A further explanation of the SWAP constraint can be found in the section SWAP: Do Not Mark Address Space Non-Swappable (see page 63). If you force the address space swappable, VSAM is forced to rebuild the expanded storage cache of CIs each time the address space is swapped back into storage. The HIPERSPACE data is not swapped out and back in with the application. If you do not want the LSR buffers page fixed, specify UNFIX.

Additionally, there may be insufficient expanded storage available to allocate as many buffers as requested. In this situation, you can obtain less HIPERSPACE buffers than requested, and you might not obtain any. Processing continues without (or with reduced) HIPERSPACE buffer backing.

LSR: Converting Non-Shared Requests to Local Shared

Resources

Note: Specifying LSR forces CA Hyper-Buf to assign an LSR buffer pool, unless SEQUENTIAL processing is the only type of processing indicated in the ACB, or if a higher precedence LEVEL SELECTOR specifies NSR. Since LSR buffering does not perform initial positioning for sequential access, do not specify LSR for mixed mode applications that do not perform their own positioning. It is safer to specify LSR on a PROGRAM or DDNAME constraint than on a CLUSTER constraint, since many different applications can access a cluster, and some of the applications may perform some sequential processing.

CA Hyper-Buf automatically promotes random-only OPENs (for example, ACBs which specify only random processing in the ACB MACRF) to use LSR buffering, unless a matching level selector specifies NSR explicitly. You should force LSR only if a predominantly random application specifies mixed-mode processing (for example, both random and sequential in the ACB MACRF) in the ACB, and performs its own positioning for all requests.

Restriction: LSR buffering does not support chained RPLs, and does not perform any positioning for sequential applications. These conditions are not detectable at OPEN time.

Do not specify LSR buffering for any applications that do not establish a position in the file before processing records sequentially. If applications that worked before CA Hyper-Buf was activated fail with positioning errors after CA Hyper-Buf is activated, make sure that LSR processing was not requested for this application, or override the LSR constraint by specifying NSR at the PROGRAM/DDNAME level.

Do not specify LSR for applications that use chained RPLs. If the file is processed randomly, and CA Hyper-Buf automatically promotes the OPEN to use LSR buffering, override CA Hyper-Buf by requesting NSR for any programs that use chained RPLs.

Default: LSR for strictly random (ACB specified) processing, else NSR.

The LSR constraint requests that CA Hyper-Buf promote NSR OPEN requests to LSR requests. This option is designed for files which are processed randomly.

There are no operands associated with the LSR control.

Prior to DFP 2, VSAM supports only one LSR buffer pool per address space. DFP 2 added LSR pools per address space.

The LSR control is not pool specific. In environments that support multiple LSR pools, the LSR constraint requests that CA Hyper-Buf select an appropriate LSR pool for this file. If you want to assign a specific LSR pool, use the SHRPOOL control instead of the LSR control. An explanation of SHRPOOL is in the section SHRPOOL: Requesting a Specific Shared Resource Pool (see page 60). If you specify both SHRPOOL and LSRPOOL, the option that is specified last in the constraint list is used.

CA Hyper-Buf does not assign LSR pools to files that cannot use LSR buffering because of VSAM restrictions. Such files include files that are being loaded ('initial load' mode or 'create' mode), files that are opened with REUSE, and so forth. In these cases, the LSR constraint is ignored by CA Hyper-Buf.

LSRFORCE: Bypassing ACB Inspection and Data Set Content

The LSRFORCE keyword can be used at the system-default level, or any other sublevel. This keyword bypasses ACB inspection and data set content (number of records), checking, and forces the data set to be considered for LSR processing. LSRFORCE does not force the data set into a pool that has already been built by a prior open. If the High Used RBA value is zero, the file is not forced into LSR. Use of the keyword is not recommended, as it can lead to abends with empty files getting placed into LSR.

MAXBUFND: Specifying Maximum Target BUFND

MAXBUFND allows you to specify the maximum allowable number of data buffers for any particular OPEN. Dynamic buffering does not exceed this value, unless this value is less than (1+strno). You specify MAXBUFND as:

```
MAXBUFND=value
```

If you do not specify MAXBUFND for any given LEVEL selector, the SYSTEM default is used.

If you do not supply a SYSTEM default MAXBUFND, or if you specify MAXBUFND=0, (2 tracks of data buffers + STRNO +1) is used as MAXBUFND.

MAXBUFNI: Specifying Maximum Target BUFNI

This parameter limits the maximum number of index buffers that are allocated for any OPEN. Dynamic buffering allocates no more than MAXBUFNI index buffers. The following shows an example:

```
MAXBUFNI=value
```

The SYSTEM default applies to any LEVEL selector that does not specify a MAXBUFNI.

If you do not supply a SYSTEM default value, or if you specify MAXBUFNI=0, (index set number) + (strno) is used as MAXBUFNI.

MAXBUFNO: Specifying Maximum Target QSAM Buffers

Default: MAXBUFNO=(230K/BLKSIZE)

The MAXBUFNO constraint specifies the maximum number of buffers that CA Hyper-Buf can assign to any OPEN request. CA Hyper-Buf tuning algorithms do not exceed this value. MAXBUFNO can be specified at any LEVEL selector.

MAXBUFSP: Specifying Maximum Target Buffer Space

Default: MAXBUFSP=128K

The MAXBUFSP constraint specifies the maximum amount of virtual storage that dynamic buffering can use to create buffers for a file. The sum of (data CI size * bufnd) + (index CI size * bufni) does not exceed this value for any particular OPEN, unless MAXBUFSP is not at least equal to ((1+strno) * (data CI size)) plus ((strno) * (index CI size)). You specify MAXBUFSP as:

```
MAXBUFSP=value
```

Value is validated by the CA Hyper-Buf OPEN intercept routine and must be at least as large as the VSAM calculated minimum allowable buffer space. If the specified value is smaller, CA Hyper-Buf increases your specified value to the VSAM minimum.

If you do not provide any MAXBUFSP value for a particular LEVEL selector, the SYSTEM default value is used.

MINBUFND: Specifying Minimum Target BUFND

Default: STRNO+1

You can guarantee that you have at least (value) data buffers by specifying a value for MINBUFND. Dynamic buffering selects at least MINBUFND data buffers, unless this value exceeds MAXBUFSP. Specify as:

```
MINBUFND=value
```

The SYSTEM default applies to any LEVEL selector that does not specify MINBUFND.

If you do not specify a SYSTEM default MINBUFND, (strno+1) is used as MINBUFND. If your MINBUFND is less than this value, it is overridden, because strno+1 is the minimum allowable value for any NSR OPEN. (LSR requires a minimum of 3 buffers for each size buffer in the pool, regardless of the number of files/index levels, and so forth.)

MINBUFNI: Specifying Minimum Target BUFNI

Default: STRNO

MINBUFNI allows you to specify a minimum number of index buffers. Each KSDS that is OPENed receives at least MINBUFNI index buffers. Specify as:

```
MINBUFNI=value
```

If any particular LEVEL selector does not specify a MINBUFNI, then the SYSTEM default MINBUFNI is used.

If you do not specify a SYSTEM default value for MINBUFNI, (strno) is used for MINBUFNI. If your specification is less than (strno), it is overridden, because the minimum number of index buffers allowed by VSAM is (strno).

MINBUFNO: Specifying Minimum Target QSAM Buffers

Default: MINBUFNO=10

The MINBUFNO constraints allow you to specify a minimum number of buffers for QSAM files. CA Hyper-Buf buffering algorithms use at least MINBUFNO buffers. This allows you to set a performance floor for all of your processing.

MINBUFNO can be specified at all LEVEL selectors.

MINBUFSP: Specifying Minimum Target Buffer Space

Default: MINBUFNI * index CI size + MINBUFND * data CI size

The MINBUFSP parameter allows you to make sure that at least MINBUFSP bytes are used for buffers. You specify MINBUFSP as:

```
MINBUFSP=value
```

If no matching LEVEL selector specifies MINBUFSP, the SYSTEM default is used.

If you do not specify a system default, MINBUFNI and MINBUFND values are used to calculate the minimum buffer space. If you have not specified either MINBUFNI or MINBUFND, the default value(s) are used for the missing specifications.

If you do specify a MINBUFSP, it is validated against the 'true' VSAM minimum of ((1+strno) * (data CI size)) plus ((strno) * (index CI size)). This is the absolute minimum allowed by VSAM and overrides your specified MINBUFSP if yours is smaller.

MSGLEVEL: Controlling Displayed Messages

Default: MSGLEVEL=0

The MSGLEVEL constraint controls which messages are displayed by CA Hyper-Buf during OPEN and CLOSE processing. You specify MSGLEVEL as a numeric quantity; for instance, MSGLEVEL=5. Only one number can be specified on each MSGLEVEL control.

The MSGLEVEL number is interpreted as follows:

0
: display only critical errors (default)

1
: BUFFERSPACE

2
: (reserved)

3
: EXCLUSIONS

4
: (reserved)

5

BYPASSES

6

(reserved)

7

SELECTIONS

8

(reserved)

9

(reserved)

10

DEBUGGING

All messages associated with MSGLEVELs less than or equal to your specified MSGLEVEL are displayed. For example, if you specify MSGLEVEL=5, all messages for levels 0 through 5 are displayed. Specifying MSGLEVEL=10 displays all possible messages.

All MSGLEVEL information is displayed by WTO and appears on the job log for each job. These messages can also appear on the operator console if not inhibited by message suppression facilities.

NONSWAP: Mark Address Space Non-Swappable

Default: SWAP, unless user specified HIPERSPACE.

The NONSWAP parameter requests that CA Hyper-Buf make the address space non-swappable. This parameter should be reserved for critical jobs since non-swappable jobs are exempt from normal SRM load balancing, and can cause overall system degradation if used indiscriminately.

The address space is not marked non-swappable until the cluster that specifies NONSWAP is opened. If that cluster is never opened, the address space is not made non-swappable. The address space is not marked swappable when the file is closed. The step remains non-swappable from the time of the OPEN until the step terminates.

The NONSWAP control has no operands. Simply add NONSWAP to any other constraints that you specify for any cluster, job, program, or program/ddname combination.

NOSTATS: Remove STATS Constraint That Results From Merge Processing

You can remove a STATS constraint that may have 'filtered down' during the merge process by specifying the NOSTATS constraint.

For more information, see the section STATS: Requesting CLOSE Statistics (see page 63).

NSR: Restricting Conversion to Local Shared Resources

Default: NSR, unless random processing specified in ACB.

The NSR control has no operands and forces CA Hyper-Buf to use NSR buffering. NSR buffering can benefit mixed-mode files (for example, both random and sequential or skip sequential) if a substantial amount of the processing is sequential. If neither LSR nor NSR is specified, the default action is as follows:

```
OPEN  FOR  SEQUENTIAL   PROCESSING  - NSR  BUFFERING
OPEN   FOR  MIXED                PROCESSING   - NSR  BUFFERING
OPEN   FOR  RANDOM            PROCESSING   - LSR  BUFFERING
```

CICS Requirements: If you request CICS files to be buffered by including the DFHSIP program, the DFHCSD, DFHRSD, DFHLCS, and DFHGCD files must be specified with NSR. IBM does not support these files in an LSR-buffered environment. The following shows a sample using NSR:

```
MODE=INCLUDE

  PROGRAM=DFHSIP

     DDNAME=DFHCSD NSR

     DDNAME=DFHRSD NSR

     DDNAME=DFHLCD NSR

     DDNAME=DFHGCD NSR
```

QSAM: Enable/Disable QSAM Buffering

Default: QSAM=YES

The QSAM constraint allows you to enable and disable QSAM buffering at any LEVEL selector. Specify as follows:

```
QSAM=x
```

where x is:

YES to enable QSAM buffering for this LEVEL

NO to disable QSAM buffering for this LEVEL

RANDOM: Overriding ACB Specified Processing Options

Restriction: RANDOM must follow a DDNAME level selector. RANDOM can be used for applications that perform mostly RANDOM processing, but specify DYNAMIC in the file definition.

Selecting DDNAME and specifying RANDOM forces the dynamic buffering routines to select buffers as if RANDOM had been specified in the file definition instead of DYNAMIC. This results in more index buffers, which should improve random accesses.

There are no operands for RANDOM.

RMODE31: Specifying VSAM Buffer and Control Block

Residency

Default: RMODE31=NONE

The RMODE31 parameter allows you to specify the residency of the VSAM control blocks and buffers. You can allocate buffers above the line and control blocks below, both above, both below, or control blocks above and buffers below. (The latter is useful for 24 bit applications that use LOCATE mode processing).

RMODE31 has an operand that designates which areas are to be allocated above 16M. The following example shows the possibilities:

```
. . .

RMODE31=CB * VSAM Control blocks above 16M

. . .

. . .

RMODE31=BUFF * VSAM Buffers above 16M

. . .

. . .

RMODE31=ALL * Both buffers and control blocks above

. . .

. . .

RMODE31=NONE * Both allocated as specified in the ACB

. . .

. . .
```

RMODE31=ALL can be specified in SYSTEM_DEFAULTS. If you make this the default, however, make sure that you include PROGRAM constraints that specify RMODE31=NONE for any AMODE 24 applications that use either LOCATE mode processing or that access VSAM control blocks directly. Failure to exempt these applications from areas above 16M will result in S0C4 abends.

If you specify RMODE31=ALL in SYSTEM_DEFAULTS and experience S0C4/5 abends in jobs that used to run successfully, add a PROGRAM level selector with RMODE31=NONE in any INCLUDE mode control group so that all areas are below 16M for this program.

IDCAMS accesses (at least sometimes) the VSAM control blocks and is (at this time) an AMODE=24 program. If you specify RMODE31=ALL in the SYSTEM_DEFAULTS, include a PROGRAM=IDCAMS RMODE31=NONE in one of your INCLUDE MODE control groups.

There is no special direct relationship between COBOL1 or COBOL2 programs and the HYPERBUF RMODE31 constraint.

If data is accessed in the VSAM buffers or control blocks directly and the constraints specify RMODE31=ALL there may be problems. There should only be problems with respect to data in the buffers if LOCATE mode processing is performed by the program. Otherwise, specifying RMODE31=ALL or BUFF in the constraints should be possible. HYPERBUF controls the location of these resources at open time and is transparent to the application. If the program attempts to look at something in the buffer or control block that is above the line and if the program is running in 24-bit mode, an SOC4 abend could occur. Test the program with each constraint to see if this happens. It does not matter if the COBOL1 or COBOL2 program is running in 24-bit mode as long as it is not accessing data in the buffer or control blocks directly. You may be able to run the program with RMODE31=ALL/BUFF.

RUNMODE: CA Hyper-Buf Mode of Operation

Default: RUNMODE=FULL

Specifications: RUNMODE=FULL RUNMODE=RESTRICTED

Restriction: This constraint can only be specified at the system-default level.

The RUNMODE keyword is used to restrict what job steps get analyzed by CA Hyper-Buf. The normal setting, RUNMODE=FULL, allows the constraint table to be applied to every job step and task step on the system in which CA Hyper-Buf is active. RUNMODE=FULL is the default.

RUNMODE=RESTRICTED

This specification forces CA Hyper-Buf to look for a specific DD statement to be in a job step or task step before the constraints are applied to the job or task. The following DD statement must be coded in each step to be analyzed if RUNMODE=RESTRICTED is active:

```
//CAHBINCL  DD   DUMMY
```

If RUNMODE=RESTRICTED is specified and the CAHBINCL DD statement is not present, CA Hyper-Buf will not do any analysis of this job/task step.

The RUNMODE=RESTRICTED parameter is implemented so that CA Hyper-Buf testing can be performed on an individual job/task basis. Such restriction might prove useful to the systems programmer while evaluating new constraints on a test system.

RUNMODE=FULL

This specification allows CA Hyper-Buf to run in the normal mode. Each job or task step will have the constraint table applied to it, looking for matches or exclusions as the table may dictate.

Excluding CA Hyper-Buf from a Step While in RUNMODE=FULL

There may be situations where it is desired to exclude CA Hyper-Buf processing from a particular step. Some examples are specialized monitor programs, such as high speed check reading applications. There could be other instances as well where you would not want the CA Hyper-Buf analyzer in the address space for a particular step. This can be accomplished by including the following DD statement in the job/task step you wish to exclude from CA Hyper-Buf analysis.

```
//CAHBEXCL  DD   DUMMY
```

On the first open of a step, the CAHBEXCL DD statement is searched for, and if found, turns off any CA Hyper-Buf processing for that job/task step.

SEQUENTIAL: Overriding ACB Specified Processing Options

Restriction: Can be specified only following a DDNAME level selector. There are some vendor supplied (and possibly in-house) applications that misrepresent the type of processing that they actually perform. This is usually the result of a general purpose I/O routine that is called by several different but related programs.

This common I/O module specifies all possible OPEN options to be ready for any possible type of request. For example, there are some vendor supplied COBOL applications that specify DYNAMIC file access in the I/O module, but one or more of the programs that call this I/O module are purely SEQUENTIAL (or purely RANDOM).

For applications that actually use SEQUENTIAL processing but specify DYNAMIC file processing, additional performance can be gained by specifying SEQUENTIAL in a LEVEL selector that matches that application.

For example, if you have purchased a vendor supplied MRP, payroll, or other package that uses SEQUENTIAL processing, but was written with DYNAMIC specified in the file declaration, you should select that application (by program name/ddname) and specify SEQUENTIAL, as shown below.

```
   . . .

. . .

PROGRAM=MRP0001

   DDNAME=ORDERS

   SEQUENTIAL

   . . . .

. . . .
```

This forces the dynamic buffering calculation to be performed as if the OPEN was purely SEQUENTIAL. This usually results in more data buffers, which improves performance for the sequential processing.

Specifying SEQUENTIAL overrides the automatic assignment of random files to LSR pools.

There are no operands for SEQUENTIAL.

SHRPOOL: Requesting a Specific Shared Resource Pool

Multiple LSR pools are supported in some environments. DFP2 pools are supported in some environments. DFP2 allows eight LSR pools (pools 0 through 7), and DFP3 allows 256 LSR pools (pools 1 through 255). DFP3 further subdivides each LSR pool into separate data and index sub-pools.

The SHAREPOOL control allows you to assign a specific LSR buffer pool to a particular OPEN request. You can group several different files into the same LSR pool, for example. You can specify which files share a pool by using the SHAREPOOL control.

The SHAREPOOL control has a single operand. The operand is the pool number that you wish to be associated with this OPEN request. The specified pool must be in the range {1,...,MAX} where MAX is the maximum pool number allowed in your environment.

Specify SHRPOOL=0 only if you want to remove a pool specific selection that results from a constraint merge, for example:

```
. . . .

. . . .

CLUSTER=A.B.C

   SHRPOOL=3,

. . .

. . .

CLUSTER=D.E.F

   SHRPOOL=3

. . . .

. . . .

PROGRAM=XYZ

   DDNAME=LOG   * Assume this DD points to Cluster A.B.C

      SHRPOOL=0 * WILL NOT use SHRPOOL 3 - pool will be chosen

                * by CA-Hyper-Buf

. . . .

. . . .
```

The example shows that clusters A.B.C and D.E.F normally share the same LSR pool (Pool 3), if both happen to be opened in the same job step. However, if PROGRAM XYZ OPENs CLUSTER A.B.C on DDNAME LOG, SHRPOOL 3 is not automatically assigned. In this case, the file is dynamically assigned an unused LSR pool. This can be any LSR pool that is not assigned to another file, including pool 3, if that pool is unassigned.

CA Hyper-Buf attempts to determine which pools are assigned for each address space at the time of the first OPEN for that address space. Any pools that do not appear to be used are eligible for assignment by the LSR control. CA Hyper-Buf assigns the highest available unassigned LSR pool. Each subsequent LSR OPEN uses a lower number pool. Pools that are explicitly assigned by SHAREPOOL are skipped in this selection process.

Note: Dynamically allocated data sets may not yet be allocated at the time of the first OPEN. Any files that are dynamically allocated after the first OPEN in an address space may be forced to use a pool other than the SHAREPOOL specified pool, if the SHAREPOOL specified pool is already allocated to another LSR file.

If all available LSR pools are already assigned, the LSR option is ignored, and the file is opened with NSR buffering.

SIS: Setting Sequential Insert Strategy

Normally, VSAM splits full CIs in half (at nearest logical record boundary) when required to make room for records that are inserted randomly into a KSDS file. This philosophy works well for truly random insertions.

Many applications randomly insert clusters of records instead of single records. These applications randomly position to a desired key in the file then insert several logical records in ascending key sequence, then randomly position and insert another cluster of records.

Since these files are opened with random access specified in the ACB, VSAM normally uses random insert strategy, which splits full CIs in half, as described above.

or an application that performs random clustered insertions, each CI split causes approximately one extra I/O for every (number of records in a CI)/2 whenever a CI split occurs. For example, if a CI holds ten records (average), then random insertion of these clustered records causes approximately one extra I/O for every five insertions. If many clustered inserts occur in the same control area, these inserts also cause CA splits, each of which causes potentially hundreds of I/Os.

Sequential insert strategy can avoid many of these CI and CA splits and the associated I/Os, if your inserts are mostly clustered, as described above.

SIS has no operands.

SMFID: Specifying SMF Recording Record ID

Default: None

The SMFID constraint allows you to specify the SMF record ID for SMF statistics recording. If you do not specify an SMFID, no SMF recording is performed.

If you select an SMFID, it must be a valid user SMF record ID. The value must be between 127 and 256. You should make sure that the number you choose does not conflict with any other SMF record IDs in your system. If you select an 'in use' SMF record ID, reporting programs will not produce desired results.

To produce SMF records, you must select an SMFID and activate SMF recording for that record ID.

SMFID can be specified anyplace in the control file, but only the last SMFID value specified is used for all SMF records. Specify SMID=0 at any level to suppress SMF recording for that specific level.

STATS: Requesting CLOSE Statistics

Default: NOSTATS

The STATS constraint requests CA Hyper-Buf to display file statistics at CLOSE time. STATS can be specified at any level. The statistics are displayed only for CLOSEs that match a level selector which specifies STATS. The statistics are displayed on the job log.

The STATS constraint requires no operands. You can specify STATS independently of MSGLEVEL. You can specify STATS to be displayed whether CA Hyper-Buf modifies the buffers for an OPEN.

You can use STATS to 'benchmark' CA Hyper-Buf performance. For example, you can run two identical job streams with two different job names. One of the jobnames should be buffered by CA Hyper-Buf; the other job name should be EXCLUDED. STATS should be specified for both jobs, or as the SYSTEM_DEFAULT. By running these 'parallel' jobs, you can see the difference in EXCP caused by CA Hyper-Buf by comparing the job logs after they complete execution.

Also see the related constraint NOSTATS in the section NOSTATS: Remove STATS Constraint That Results From Merge Processing (see page 55).

SWAP: Do Not Mark Address Space Non-Swappable

Default: SWAP, unless user specified HIPERSPACE

Normally, CA Hyper-Buf automatically asserts NONSWAP if HIPERSPACE is requested. If you want HIPERSPACE backing of buffers but do not want the address space marked non-swappable, you can specify SWAP in addition to HIPERSPACE. See the description of HIPERSPACE in the section HIPERSPACE: Backing LSR Buffers in HIPERSPACE (see page 47) for further information.

The SWAP constraint can also be used to override a NONSWAP constraint that is in effect from a higher LEVEL constraint merge. SWAP does not actually force the address space swappable. It merely inhibits CA Hyper-Buf from marking the address space non-swappable. Jobs that are non-swappable because of external factors (such as being named in the PPT) are not affected by the SWAP constraint.

There are no operands associated with the Constraint SWAP control.

SYSTEM_DEFAULTS - Establishing Default Parameters

All of the constraints can be assigned a default value. The default value is the value that is used for any constraints that are not specified on a matching level selector. These default values are assigned in the SYSTEM_DEFAULTS section of the GVBDBFON SYSIN file. The SYSTEM_DEFAULTS section begins with the SYSTEM_DEFAULTS command. This command can be abbreviated as SYSTEM.

All of the constraints (except for the above noted restrictions) can be specified in the SYSTEM_DEFAULTS section. Any constraint values assigned in this section will become the default values for those constraints. Any constraints that you do not assign a default value will have a value assigned to them by the OPEN intercept routine. The OPEN assigned values are discussed under each of the constraint topics earlier in this manual.

TEST: Forcing TEST Mode

When TEST mode is asserted in the constraints, CA Hyper-Buf performs all buffering calculations, but does not actually implement the calculated buffer space. The calculated values are displayed if MSGLEVEL is 1 or greater.

The calculated buffers are displayed following message GVB668I, which identifies the values as being TEST values. These values are only displayed and not implemented for this OPEN. Normal VSAM buffering is in effect, and the buffers are determined strictly from AMP, ACB, and catalog parameters by VSAM, as though CA Hyper-Buf was not active at all.

TRACKS: Specify Multi-track Buffering

Specify this value as TRACKS=n, where n is the number of tracks, up to 1 cylinder. The TRACKS constraint forces CA Hyper-Buf to allocate enough buffers to hold the number of tracks specified in n. This constraint can be used for both QSAM and VSAM.

Note: The TRACKS value can cause CA Hyper-Buf to exceed MAXBUFSP/NO/NO.

UNFIX: Keeping LSR Buffers and IOBs Pageable

Restriction: Applies only to LSR buffers. See the FIX restrictions in the section FIX: Page Fixing LSR Buffers and IOBs (see page 46) for further information.

Specify the UNFIX constraint to override the fixing of buffers for HIPERSPACE or LSR buffering or to override a FIX constraint that results from the constraint merge process.

See the following control stream for an example:

```
  CLUSTER=PAYROLL.MASTER.FILE

     SHRPOOL=5 *  Could  also be  LSR

     FIX

   . . .

. . .

   PROGRAM=PAYROLL

     UNFIX
```

In this example, any program that opens PAYROLL.MASTER.FILE will have page fixed buffers for that cluster. The only exception is if the cluster is opened by PROGRAM PAYROLL, in which case the buffers are not page fixed.

USE: Invoking a Constraint Prototype

The USE keyword allows you to invoke a previously defined constraints prototype. The format of the USE keyword is:

```
(any   level selector or  DEFINE)

(optional  constraints)

USE=name

(optional  overrides of  prototype values)
```

The *name* variable must select a prototype that has been previously defined. The USE keyword is equivalent to specifying all of the constraints that are specified in the named prototype definition.

The following example shows a prototype definition and two invocations of the defined prototype:

```
SYSTEM_DEFAULTS

   (any  defaults)

MODE=INCLUDE

DEFINE=PRODZ     *  Define   prototype named  PRODZ

   MAXBUFND=60 MINBUFND=25
   LSR
   MINBUFNI=20 MAXBUFNI=50
   HIPERSPACE    DEFER  SHRPOOL=3
   MSGLEVEL=6
   ...
   ... (other constraints)
   ...

JOB=PAYROLL
   USE=PRODZ   *Invoke  all PRODZ  constraints
   MAXBUFND=80 *Override PRODZ   MAXBUFND

PROGRAM=XYZ+
   USE=PRODZ   *Invoke  prototype PRODZ  with  no  overrides

...
```

Chapter 4: ISPF Dialogs

You can use the ISPF dialogs to monitor and control CA Hyper-Buf. Each dialog is described in detail in this chapter.

Chapter 5, "Batch Utility Programs" describes the batch utilities that perform the same functions as these dialogs. You can use either the batch utilities or the dialogs.

This section contains the following topics:

CA Hyper-Buf Primary Menu

```
------------------CA-Hyper-Buf for MVS  Version 11.05.00-------------09:27

OPTION   ===>

  CA-Hyper-Buf  is  currently  ACTIVE

I      Info     - Display information about   CA-Hyper-Buf

C      Constraints - Show currently active constraints

M      Model    - Model using current constraints

Authorized  functions:

A      Activate - Activate, Refresh, or  Validate

D      Deactivate  - Deactivate CA-Hyper-Buf

Enter  END command   to return to the   previous menu or  X  to exit

Copyright (c)  2011  CA.
```

Menu Options

This menu is the main entry point into the CA Hyper-Buf ISPF dialogs.

From this menu, you can select:

I

Displays information about the currently installed version of CA Hyper-Buf. A complete description of this function is included in the description of the information panel.

C

Displays the constraints that are currently active on your system. If CA Hyper-Buf is not active, no constraints are shown. Further details are available in the description of the ACTIVE CONSTRAINTS panel.

M

Models user specified combinations of selectors and displays the resulting composite constraints. Further information is available in the description of the composite constraints panel.

A

Validates, activates, or refreshes constraints. Full details are supplied in the description of the ACTIVATE panel. (Authorized function.)

D

Deactivates CA Hyper-Buf after requesting confirmation. (Authorized function.)

Information Panel: Display Current Release Information

```
------------------CA-Hyper-Buf for MVS  Version 11.05.00------------------

OPTION   ===>

       Current Status   -       ACTIVE  Loaded  at 80AF3D98

       Initialized on  FRI,  23  JAN  04  at 16.24.35 by  GVBDBFON Refreshed on  FRI,
23     JAN  04  at  16.24.35 by  GVBDBFON

       Csect    Ver     Date    Time    Loaded  from.  . . .  .

       GVBDYNBF 110000  040106  1233    (CSA)

       GVBDYN2  110000  040106  1233    CAI.CBS3LINK

       VBDBFON  110000  040106  1233    CAI.CBS3LINK

       GVBDBOFF 110000  040106  1233    CAI.CBS3LINK

       GVBREFR  110000  040106  1235    CAI.CBS3LINK

**************************  BOTTOM  OF   DATA  ********************************
```

Panel Options

This panel is displayed when you select option I from the main menu. The following information is displayed on this panel:

1. The current status of CA Hyper-Buf on this system. This field displays either ACTIVE or INACTIVE. In this sample panel, CA Hyper-Buf is currently ACTIVE.
2. The address of the OPEN intercept routine. This field is displayed only if CA Hyper-Buf is currently ACTIVE.
3. The date and time CA Hyper-Buf was initialized, and the date and time of the most recent REFRESH.
4. The assembly date and time for several key CA Hyper-Buf CSECTs. If you call technical support with any questions, they may ask you for this information.
5. Where each module was found. This allows you to verify that no old modules are found in libraries concatenated ahead of the desired release modules. If CA Hyper-Buf is active, module GVBDYNBF shows that it was loaded from (CSA). In this case, the assembly date and times are extracted from the currently active OPEN intercept module that is loaded in CSA. The actual library that this module was loaded from is not displayed.

Constraints Panel: Displaying the Currently Active Constraints

```
------------------CA-Hyper-Buf for MVS  Version 11.05.00------------------

OPTION   ===>

   Active  constraints: SYS_DEFAULTS

   MAZBUFSP=128K

       DFLTMODE=INCLUDE

       ACBOVR

       MSGLEVEL=10

       STATS

       HIPERSPC=1

       DYNREG=YES

   DEFINE=DRAT

       MAXBUFSP=500K

       MSGLEVEL=10

       STATS

       ACBOVR

       AMPOVR=MAX

       MINBUFNI=27

   SCLASS=SPECIAL
```

Panel Description

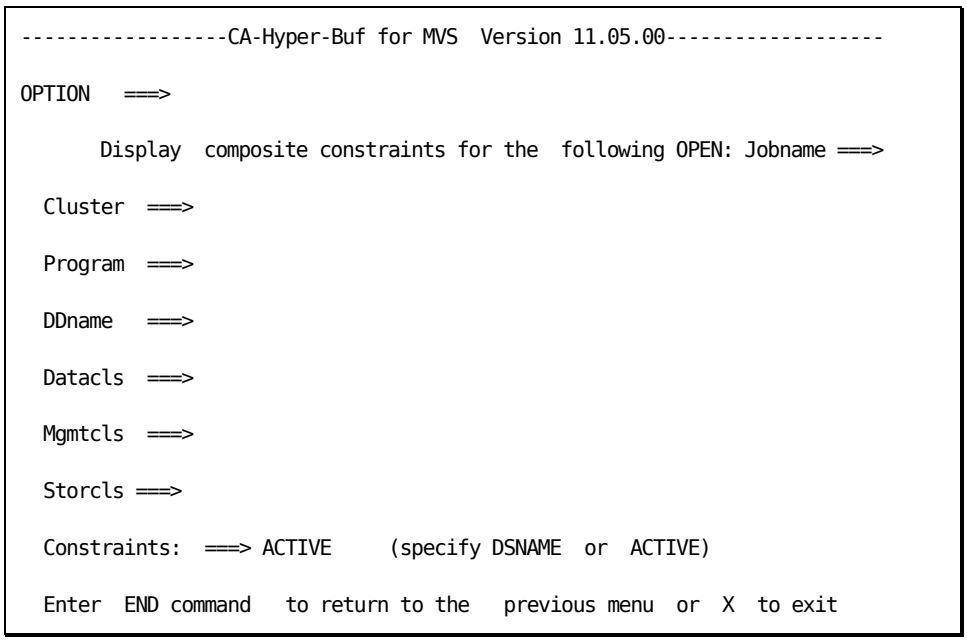
This panel is displayed when you select option C from the main menu. The currently active constraints are displayed only if CA Hyper-Buf is active.

These constraints are displayed in the same order that CA Hyper-Buf searches the constraints at OPEN time. This is generally not the order in which the constraints were read at initialization time. Also, since the constraints are extracted from CA Hyper-Buf control blocks, your comments do not appear in this output.

Unless you have a very small constraints file, more than one screen is required to display all of the constraints. You can scroll through all of the constraints by using the ISPF scroll keys.

Level Selectors that are EXCLUDED are shown with an 'X' in the first column of the display.

Modeling Panel: Specifying Level Selectors



Panel Description

This panel allows you to model specific level selectors. You can specify selected JOB, CLUSTER, PROGRAM, DD, and SMS class information for the modeling routine. The modeling program compares these parameters against the constraints file, and displays the composite constraints for this combination of level selectors.

If you supply a cluster name, but do not specify any SMS class information, the modeling program obtains the SMS class information for that cluster from the catalog, and uses those values in determining the composite constraints. If you specify any SMS parameters, your specifications will override the corresponding values obtained from the catalog.

You can also supply the name of a file that contains the constraints to be used for the modeling process. You specify this file in the Constraints: ===> input field.

If you specify ACTIVE in the constraints input field, the modeling process uses the currently active constraints. In this case, CA Hyper-Buf must be active on your system. The resulting composite constraints are the same as those generated by the OPEN intercept routine for the same combination of level selectors.

If you supply a file name in the constraints input field, the modeling process uses that file to determine the composite constraints. This does not affect CA Hyper-Buf in any way. The constraints file specified here is used only for the modeling process.

Sample Panel 1: Following is an example of a completed Modeling Parameters Selection panel.

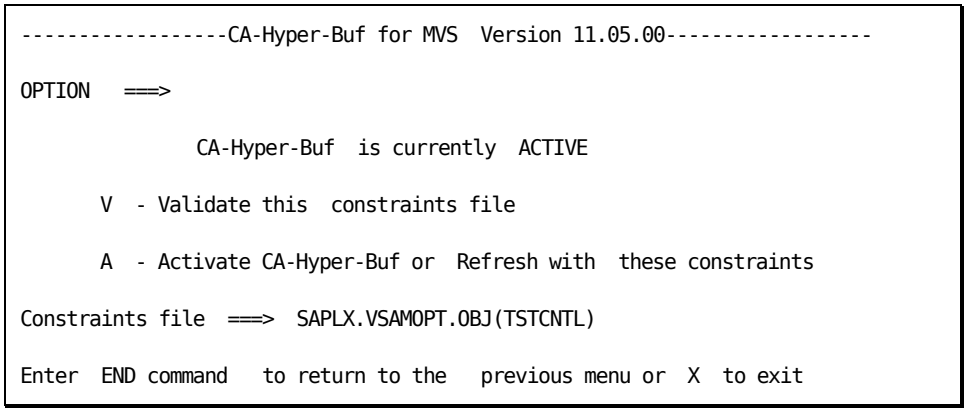
This example selects JOB AFS32762, executing PROGRAM GVRESTOR, opening CLUSTER SAPLX.TEST.CLUSTER on DDNAME FILEIN. When you press Enter, CA Hyper-Buf calculates and displays the composite constraints for this OPEN.

```
------------------CA-Hyper-Buf for MVS  Version  11.05.00------------------

OPTION   ===>

       Display  composite constraints for the  following OPEN:

       Jobname ===> AFS32762

       Cluster  ===> SAPLX.TEST.CLUSTER

       Program  ===> GVRESTOR

       DDname      ===> FILEIN

       Datacls  ===>

       Mgmtcls  ===>

       Storcls ===>

       Constraints:  ===> ACTIVE(specify DSNAME  or  ACTIVE)

       Enter  END command   to return to the   previous menu  or  X  to exit
```

Sample Panel 2: This panel displays the output produced from the previous panel using composite constraints.

```
---------------- CA-Hyper-Buf for MVS  Version 11.05.00  ---- Row  1  to 1

OPTION      ===>
       SCROLL=

       Composite  constraints:

                MAXBUFSP=384K

                MAXBUFND=20

                MAXBUFNI=30

                MINBUFSP=32K

                MINBUFND=10

                MINBUFNI=15

                CUSHION=200K

                DFLTMODE=INCLUDE

                ACBOVR

                MSGLEVEL=10

                STATS

                DEFER

                RMODE31=ALL

                TRACKS=16

                QSAM=YES

                RUNMODE=RESTRICTED

******************************  Bottom  of  data ***********************
```

Activate Selection Panel: Validate, Activate, or Refresh



Panel Description

This panel performs authorization checking to determine if you are authorized to perform this function.

This panel is displayed when you select A from the main menu.

You can specify a constraints file in the Constraints file ===> input field, then select 'V' to validate, or 'A' to activate (if CA Hyper-Buf is not currently active) or refresh (if CA Hyper-Buf is currently active).

If there are any errors in the constraints file, the activate/refresh is not successful. In this case, the following screen is displayed. (This is the same screen that is displayed by the validate function).

```
------------------CA-Hyper-Buf for MVS  Version  11.05.00-------------ROW  1  OF
204

OPTION   ===>

       Current  status  -  ACTIVE

       Constraints file  - SAPLX.VSAMOPT.OBJ(TSTCNTL)

          Returned -    000

...:....1....:....2....:....3....:....4....:...5....:....6....:....7....:

NOSTAE

   SYSTEM_CONSTRAINTS

       MSGLEVEL=10

       STATS

       ACBOV

   DEFAULTMODE=INCLUDE

   MODE=INCLUDE

   MCLASS=FRSTCLAS

       MAXBUFNI=42 MINBUFNI=30

       MAXBUFND=60 MINBUFND=50

       MAXBUFSP=1024K
```

Chapter 5: Batch Utility Programs

This section contains the following topics:

GVBVALD—Validate CA Hyper-Buf Control Stream

GVBVALD is a utility program that validates your control statements. You can run this job to check the validity of your defaults, level selectors, and constraints before you initialize CA Hyper-Buf on your machine.

Sample JCL member GVBVALD is provided in the CAISAMP library.

Note: The CNTLFILE DD may point to any 80-byte LRECL, RECFM=FB data set that contains the control statements that you want to validate.

Output contains a listing of the control statements, along with any error messages that may be produced. The return code is set to non-zero if any errors occur.

Note: GVBVALD must be executed from an APF authorized library.

GVBDBFON—Activate CA Hyper-Buf

GVBDBFON is the batch program that initializes CA Hyper-Buf.

Sample JCL member GVBDBFON is provided in the CAISAMP library.

Note: The DD statements for the first step are described earlier in the section concerning GVBVALD.

The DD statements for the GVBDBFON step are as follows:

- CNTLFILE points to the control stream that was validated in the VALD step.
- SYSPRINT for output messages.

In this example, GVBVALD is placed in front of GVBDBFON to make sure that there are no errors in the control statements. If GVBVALD detects any errors, GVBDBFON is skipped, ensuring that the controls supplied to GVBDBFON are error free.

Starting GVBDBFON with an invalid control stream does not cause any damage to your system since GVBDBFON also validates the control statements. However, an invalid control statement(s) can cause some minor fragmentation of CSA. Therefore, you should always run GVBVALD to validate input to GVBDBFON.

GVBDBFON reads and validates the control statements as it builds the constraint table in CSA and then loads the OPEN intercept into CSA. From that point in time until you deactivate CA Hyper-Buf, all VSAM OPENs are screened by the CA Hyper-Buf OPEN intercept routine. Any OPENs that match your level selectors are dynamically buffered or excluded, as described earlier in this manual.

GVBDBOFF—Deactivate CA Hyper-Buf

This program deactivates CA Hyper-Buf. You can run this program at any time to remove CA Hyper-Buf from your system.

Sample JCL member GVBDBOFF is provided in the CAISAMP library.

Note: GVBDBOFF requires no input.

After you run this job, OPENs are no longer screened by the OPEN intercept routine.

You can restart dynamic buffering at any time after GVBDBOFF completes by running GVBDBFON again, or by the ACTIVATE ISPF panel, if you are authorized to do so.

GVBREFR—Refresh Constraints

This program changes the constraints in effect without stopping and restarting CA Hyper-Buf. Supply the new constraints to GVBREFR using the SYSIN file. Normal constraint validation and verification are performed by GVBREFR.

Sample JCL member GVBREFR is provided in the CAISAMP library.

Note: As with GVBDBFON, you should validate the constraints prior to invoking GVBREFR to avoid CSA fragmentation.

GVBSTAT—Display Current Status of CA Hyper-Buf

This program displays the current status of CA Hyper-Buf, and displays assembly date/times for key CA Hyper-Buf modules.

If CA Hyper-Buf is active when GVBSTAT runs, the address of the OPEN intercept routine, the time that CA Hyper-Buf was initialized, the time of the last (most recent) refresh, and other key information is displayed. The assembly date and time for GVBDYNBF is taken from the copy that is currently loaded in the CSA, if CA Hyper-Buf is already active. All other modules are loaded from the JOBLIB/STEPLIB/LINKLIST to determine the assembly date and times, and the results are displayed.

If CA Hyper-Buf is not active when GVBSTAT runs, all modules are loaded to determine the assembly date and time.

Sample JCL member GVBSTAT is provided in the CAISAMP library.

Note: No input control statements are required for GVBSTAT. You can test the release levels of multiple copies of CA Hyper-Buf by adding a JOBLIB/STEPLIB that points to the library to be tested. If you do not supply JOBLIB/STEPLIB, or if the modules do not reside in the JOBLIB/STEPLIB libraries, the first copy of each module in LINKLST is tested/displayed.

Output is displayed on the SYSPRINT DD. GVBSTAT sets the return code to 8 if CA Hyper-Buf is not active. This value can be tested by the COND= parameter on subsequent job steps.

GVBISIT—Determine if CA Hyper-Buf is Active

Program GVBISIT detects CA Hyper-Buf if it is active on your system. You can use GVBISIT to verify that CA Hyper-Buf is active for jobs that require CA Hyper-Buf, and that it is inactive for any jobs that you want to run when CA Hyper-Buf is down.

GVBISIT does not require any input or output files. The only output is the return code passed back to OS/390 in R15. If CA Hyper-Buf is inactive when GVBISIT runs, the return code in R15 is set to zero. If CA Hyper-Buf is active, R15 is set to eight.

For example, you may decide to remove all of the old release CA Hyper-Buf modules before you install a new version. This is only safe to do if CA Hyper-Buf is not active, since OPEN processing when CA Hyper-Buf is active LOADs some of these modules, and removing them while CA Hyper-Buf is active can cause your SYS1.DUMP data sets to fill up (your jobs still run, but each OPEN triggers another dump).

In this case, you can insert GVBISIT at the beginning of the clean-up job and use the return code to 'skip' the delete step if CA Hyper-Buf is active.

If you have a job that runs too long without CA Hyper-Buf active, you can use the same return code to skip the step if CA Hyper-Buf is not active and issue the appropriate messages. This way, you can make sure that the job runs only if CA Hyper-Buf is active.

Sample JCL member GVBISIT is provided in the CBS3JCL library.

GVREPORT SMF Reporting Utility

The GVREPORT SMF Reporting utility is designed as an aid to clients in the planning, implementation, and tuning of the files that are selected to be buffered by CA Hyper-Buf. Seven reports are provided that cause most tasks to be accomplished readily. However, GVREPORT is designed to allow customized reports to be created.

The following reports are provided as a set of tools to begin the reporting and tuning process:

- JOB INFORMATION
- DATA COMPONENT
- LSR DATA COMPONENT
- LSR INDEX COMPONENT
- NSR INDEX COMPONENT
- MISC FIELDS

Clients can alter the reports at anytime to build a customized report to satisfy individual site requirements.

Reports are defined using simple field title constructs such as the following command statements:

- Report
- Title
- Subtitle
- Field
- Fields
- Select

Samples are provided in the CBS3OPTN data set on the install tape. The member name is GVREPORT. The seven reports are stacked in a single member and require relocation to separate members before they can be executed.

Report Fields

SMFJOBNM

Job name

SMFSTPNM

Step name

SMFPRGNM

Program name

SMFCDENM

CDE Program name

SMFDDNM

DDNAME

SMFXXDTE

SMF date

SMFXXTME

SMF time

SMFMRFU

User's ACB MACRF

SMFMRFH

Hyper-Buf ACB MACRF

SMFDCISZ

Data CISIZE

SMFDBFNU

Users BUFND

SMFDBFNH

Hbufs BUFND

SMFDEXCP

Data component EXCPS

SMFDPXCP

Data component path EXCP

SMFRETRV

Records retrieved

SMFINSER

Records inserted

SMFDELTS

Records deleted

SMFUPDTS

Records updated

SMFCISPL

CI splits

SMFCASPL

CA splits

SMFLSDLA

LSR lookasides - Data

SMFLSDBR

LSR buffer reads - Data

SMFLSDNW

LSR non user writes - Data

SMFLSDUW

LSR user writes - Data

SMFLSDNM

LSR pool ID - Data

SMFLSDHB

Hyperspace buffers - Data

SMFLSIEX

Index EXCP count - LSR

SMFLSIPX

Index path EXCP count - LSR

SMFLSIBU

USERS BUFNI - LSR

SMFLSIBH

HBUFS BUFNI - LSR

SMFLSICI

Index CISIZE - LSR S

MFLSILA

Index lookasides - LSR

SMFLSIBR

Index buffer reads – LSR

SMFLSINW

Index non user writes - LSR

SMFLSIUW

Index user writes - LSR

SMFLSINM

Index LSR pool ID

SMFLSIHB

Hyperspace buffers - Index

SMFIBFNU

User BUFNI - NSR

SMFIBFNH

HBUF BUFNI - NSR

SMFICISZ

Index CISIZE - NSR

SMFIEXCP

Index ECXPs - NSR

SMFIPXCP

Index path ECXPs - NSR

SMFXXSID

MVS system ID

The SMFXXRTY in the constructs example is the hex value of the value in the SMFID= parameter defined in the CA Hyper-Buf Constraint Table. (See the section SMFID: Specifying SMF Recording Record ID (see page 62))

```
SELECT    SMFXXRTY EQ  X'81' (Example  only)
```

The X'81' equates to 129. (Example only)

SMFHBREL

CA Hyper-Buf release from which the SMF data was collected. The SMFHBREL in the construct example is the release level of CA Hyper-Buf.

```
SMFHBREL   EQ  X'00110000'
```

This translates to r11

SMFFLAG2

Include/exclude flag

SMFMRFU1

USERS ACB MACRF1

SMFMRFU2

USERS ACB MACRF2

SMFMRFU3

USERS ACB MACRF3

SMFMRFH1

HBUFS ACB MACRF1

SMFMRFH2

HBUFS ACB MACRF2

SMFMRFH3

HBUFS ACB MACRF3

SMFCLTYP

Cluster type

BUFSPACE

Buffer space

INRECNUM

Input record number

RECLEN

Input record length

SMFDCISZ

Data CISIZE

SMFLSICI

Index CISIZE - LSR

SMF Record Description

The layout of the SMF data that CA Hyper-Buf creates is documented in member GVBSMFR in the CBS3SAMP library. You can use this macro to map the fields into an assembler program. Before you assemble your program, copy the macro to an appropriate macro library, or copy the macro in its entirety to the front of the program you are going to assemble.

Feature Implementation Guidelines

To implement this feature, you must first update the CONSTRAINT file used for CA Hyper-Buf startup. Update the SMFID= field with the SMF record type selected for CA Hyper-Buf by your installation. The reports can be processed once the SMF records are collected.

Sample JCL member GVREPJOB is provided in the CBS3JCL library.

JCL Field Descriptions

SMFIN

The data set that contains the SMF records pulled from SMF data using the record type indicated in the SMFID parameter. The value of the SMFID parameter is a user selected value that must be over 127. This field is detailed in the in the section SMFID: Specifying SMF Recording Record ID (see page 62).

```
SMFID=129  (Example  only)
```

SMF records are generated for each VSAM OPEN and CLOSE intercepted by CA Hyper-Buf. This generates a lot of SMF records and should be monitored closely. Once a sufficient record collection has been determined, copy the SMF records from the SYS1.MANx file to a file that can be used by CA Hyper-Buf. Using IBM's IFASMFDP utility is acceptable but any SMF copy utility will function as long as the record format is not altered.

GVRSYSIN

This DD is used to reference the data set that contains the report constructs. Member GVREPORT in the CBS3OPTN library contains seven preset reports.

The SMFXXRTY in the constructs example is the Hex value of the value in the SMFID= statement in the CA Hyper-Buf Constraint Table.

```
SELECT    SMFXXRTY EQ  X'81' (Example  only)
```

The X'81' equates to 129. (Example only)

GVREPORT Sample Report Construct

The following is a working report sample. There are seven separate reports in the GVREPORT sample provided in the CBS3OPTN library. Each report should be broken out into separate reports.

```
OPTIONS   DATEFORM(MM/DD/YY) TIMEFORM(HH:MM:SS)   UPPERCASE  SORTNAME(SORT)  -

           TRACE

TITLES ('HYPERBUF   SMF REPORT - JOB  INFORMATION')

REPORT GVBSMF42  INFILE(SMFIN)

FIELDS INRECNUM  SMFFLAG1   -

        SMFJOBNM    SMFSTPNM   SMFPRGNM  SMFCDENM  SMFDDNM  -

        SMFXXDTE    SMFXXTME  SMFMRFU     SMFMRFH

FIELD   SMFDSN  (LENGTH(41))

SELECT    SMFXXRTY EQ  X'81' AND  SMFHBREL   EQ  X'00110000'

RUN
```

The RUN statement engages the report. The RUN statement is the key to running the previous report constructs. Once a RUN statement is discovered, no other statements are processed.

- The SMFXXRTY EQ X'81' denotes SMF record type 129 (Example only)
- The SMFHBREL EQ X'00110000' denotes the current version of CA Hyper-Buf executing on the system the SMF records were cut.

Continuation characters are required when continuing a selected field construct to a new line as seen in the following example.

```
FIELDS INRECNUM  SMFFLAG1   -

       SMFJOBNM    SMFSTPNM   SMFPRGNM  SMFCDENM  SMFDDNM  -

       SMFXXDTE    SMFXXTME  SMFMRFU     SMFMRFH
```

The FIELDS criteria construct has several report fields selected. The (-) dash at the end of each line denotes a continued field. Once the dash is no longer used, the field construct is complete.

The TITLE field construct is taken as a literal. Any characters or spaces within the parenthesis and quotes are considered a literal and simply placed into the report heading as is. No continuation characters are used for this construct.

The SELECT construct allows simple logic choices to be made. An example of this type of logic is looking for version specific information and the correct SMF record type.

```
SELECT    SMFXXRTY EQ  X'81' AND  SMFHBREL   EQ  X'00110000'
```

The field SMFXXRTY EQ X'81' notifies GVREPORT to select only those SMF records that match SMF record type X'81' (TYPE 129-Example only).

The field SMFHBREL EQ X'00110000' notifies GVREPORT to select only those SMF records that have a CA Hyper-Buf r11 identifier.

Since the AND logic is used, the SMF record selected must be TYPE 129 (Example only) and created by CA Hyper-Buf r11 or the record is passed up for selection.

Note: No ANDOR logic exists for GVREPORT at this time.

GVBDB001—Disable or Enable CA Hyper-Buf

There might be times where you wish to suspend CA Hyper-Buf processing, and then resume processing later. GVBDB001 facilitates this without unloading CA Hyper-Buf from memory. GVBDB001 is a

batch utility program that disables CA Hyper-Buf without removing it from the system. Use the "disable" parameter on the execute statement to suspend processing. Once disabled, the "enable" parameter on the execute statement resumes CA Hyper-Buf processing.

Sample JCL member GVBDB001 in the CBS3JCL library illustrates the use of the program.

Note: The directive for GVBDB001 is supplied via parameter information on the execute statement. "PARM=DISABLE" suspends processing while retaining the intercepts and constraints file in memory. "PARM=ENABLE" will resume CA Hyper-Buf processing. The SYSPRINT DD statement is required for any messages. If any errors occur, a non zero return code will be returned.

Chapter 6: Sample Control Streams

This section contains the following topics:

Specifying Defaults Only

The following control stream specifies only the system default values. CA Hyper-Buf attempts to optimize all OPENs within these default constraints.

```
SYSTEM_DEFAULTS         *  begin  defaults

       MAXBUFSP=100K            *  maximum  per  file

       MAXBUFND=12              *  maximum  per  file

       MAXBUFNI=8               *  maximum  per  file

       CUSHION=100K             *  leave this  much  after  OPEN

       DEFAULTMODE=INCLUDE      *  all  included
```

The above sample control stream restricts the maximum buffer space for any single file to 102400 bytes (100K). Also, the maximum allowable number of data buffers is 12, and the maximum allowable number of index buffers is 8. Since no minimums are specified, the system values are chosen as the minimums: (strno) index buffers, (strno+1) data buffers, and the default MINBUFSP value is the amount of buffer space required by MINBUFNI index buffers and MINBUFND data buffers.

All OPENs use these same rules since DEFAULTMODE=INCLUDE. The only OPENs that do not abide by these rules are those which cannot be buffered by CA Hyper-Buf. See the section Method of Operation (see page 26) for a description of excluded OPENs. Since there are no EXCLUDE mode controls, there are no exclusions.

Adding Specific Selection Criteria

In this example, notice that the control statements are free form. Some of the constraint controls are specified on the same line or on the same logical record as their owning level selector. Others are specified on separate lines. Some have a single constraint per line; others have multiple constraints on the same line. All of these specifications are valid.

```
SYSTEM *  begin  defaults

       MAXBUFSP=100K           *  maximum  per  file

       MAXBUFND=12             *  maximum  per  file

       MAXBUFNI=8              *  maximum  per  file

       CUSHION=100K            *  leave this  much  after  OPEN

       DEFAULTMODE=INCLUDE     *  all    included

                               *  unless  excluded

CLUSTER=PAY+

       MAXBUFSP=100K    MAXBUFND=22     ***  override defaults

PROGRAM=PAYBACK    MAXBUFSP=100K

       CUSHION=150K

       DDNAME=INMSTR  MAXBUFND=40

JOB=P56673

       MAXBUFSP=32K    MAXBUFND=5

CLUSTER=PAYROLL.MASTER.FILE

       MAXBUFNI=12

MODE=EXCLUDE           *  the   following are

                       *  exempt  from  buffering

       JOB=A2276

       PROGRAM=DFH+
```

In addition, notice that the JOB, CLUSTER, and PROGRAM level selectors are specified in whatever order is convenient for you. The order that the constraints are checked is determined by the defined hierarchy of the LEVEL SELECTORS, not by the order in which you specify the selectors. You can group all CLUSTER levels together, or group all levels pertaining to a particular application system together. The composite constraint table is the same in either case.

Moreover, since the LEVEL SELECTORS are independent entities, the CLUSTER=PAYROLL.MASTER.FILE level selector constraints apply to any job or program that opens this cluster (assuming that no explicit EXCLUDE is coded); not just to JOB=P56683.

This control stream specifies the same defaults as the previous example. However, the clusters, jobs, and programs that are listed before the MODE=EXCLUDE card are dynamically buffered (unless accessed by job A2276 or programs that start with DFH, which are the only EXCLUDE MODE selectors). The overrides specified with the selectors will override the default constraint values when matched at OPEN time.

If cluster PAYROLL.MASTER.FILE is opened by job X33876 which executes program H55667, MAXBUFNI is 12 (override), MAXBUFSP is 60K (default), and MAXBUFND is 12 (default). These are the limits honored by CA Hyper-Buf at OPEN time.

In another example, assume that job A2276 or program DFHSIP OPEN this same file. In this case, no dynamic buffering occurs. Both the job and program names match an EXCLUDE mode control group entry, which takes precedence over all other controls.

Finally, if job P56673, program PAYBACK, OPENs PAYROLL.MASTER.FILE on the INMSTR DD card, the following occurs:

- The composite constraint table is initialized with the system default values
- MAXBUFSP=100K, MAXBUFND=12, MAXBUFNI=8, CUSHION=100K.
- Cluster selectors are compared against the cluster name. A match occurs, so any specified constraints are overridden. Result: MAXBUFSP=100K, MAXBUFND=12, MAXBUFNI=12, CUSHION=100K.
- JOB selectors are compared against the job name. Again, a match. Result: MAXBUFSP=32K, MAXBUFND=5, MAXBUFNI=12, CUSHION=100K.
- The PROGRAM selectors are compared against the program name. Again, a match. Result: MAXBUFSP=100K, MAXBUFND=5, MAXBUFNI=12, CUSHION=150K.
- The DDNAME selectors are compared against the ddname. Result: MAXBUFSP=100K, MAXBUFND=40, MAXBUFNI=12, CUSHION=150K.

The above example is not intended to demonstrate numbers that you might want to use in your controls. These numbers were chosen to demonstrate the process that CA Hyper-Buf uses to set the limits that are in effect for any particular OPEN. You can specify any values that you choose for any of the jobs, programs, and clusters in your shop.

Sample Test Controls

The sample test control shown below is supplied in the CBS3OPTN library as member GVBSTART. You can modify these controls to test your installation of CA Hyper-Buf.

```
SYSTEM_CONSTRAINTS        *  THESE  WILL  APPLY  IF

       MAXBUFSP=200K      MINBUFSP=60K   *  NO OVERRIDES  OCCUR

       MAXBUFND=75              MINBUFND=10      *  IN  THE  FOLLOWING

       MAXBUFNI=25                MINBUFNI=6     *  CONTROL  STATEMENTS

       CUSHION=100K     *  LEAVE  100K FREE
       STATS    *  ISSUE CLOSE STATS

       RMODE31=NONE     *  DON'T  MOVE  BUFFS,  CBS
                *  ABOVE  16M, UNLESS
                *  SPECIFICALLY   REQUESTED
                *  BELOW   (ACTUALLY,    THIS
                *  IS  THE  DEFAULT  IF
                *  NOTHING  SPECIFIED)
       MSGLEVEL=1       *  ISSUE OPEN  MESSAGE
       DFLTMODE=EXCLUDE *  IF  NOT  IN INCLUDES,
                *  DON'T   DO  ANYTHING
*********     END  DEFAULTS   *****************

MODE=INCLUDE    *  START  INCLUDE   GROUP
       *
JOB=ABC+        *  ABC???   JOBNAMES WILL
       STATS    *  BE  BUFFERED  WITH
       MSGLEVEL=10      *  ABOVE  'DEFAULT'S
                *  WITH CLOSE  STATS
                *  AND  'FULL'  MESSAGES
                *
JOB=TEST+       *  ALL 'TESTXXXX'   JOBS
       MAXBUFSP=120K    MAXBUFND=15      *  WILL  BE  BUFFERED
       MAXBUFNI=20  STATS       *  WITH SELECTED  OVERRIDES
       MSGLEVEL=10      *
                *
*******     END  'INCLUDE' MODE  CONTROL   GROUP   *****

MODE=EXCLUDE    *  BEGIN 'EXCLUDE'   MODE
       *  CONTROL  GROUP
PROGRAM=DFH+    *  IF  YOU  INCLUDE   DFHSIP,
       *  ADD   A  DDNAME  FOR DFHCSD,
       *  DFHRSD,   DFHLCS,    AND DFHGCD
       *  WITH  THE  NSR  CONSTRAINT
```

These controls cause all jobs to be excluded except for jobs that have names that begin with the letters ABC or TEST. This is because the default mode is EXCLUDE, and the INCLUDE mode control group specifies generic job names ABC+ and TEST+.

Programs that have names that begin with the characters DFH are excluded. This is true even if the job name begins with ABC or TEST.

Jobs that begin with the characters ABC have the following constraints at OPEN time:

- MAXBUFSP=200K MINBUFSP=60K MAXBUFND=75
- MINBUFND=10
- MAXBUFNI=25
- MINBUFNI=6
- CUSHION=100K
- RMODE31=NONE (all from defaults) STATS (from both default and JOB) MSGLEVEL=10 (from JOB level selector)

Jobs that begin with the characters TEST have the following constraints at OPEN time:

- MINBUFSP=60K MINBUFND=10
- MINBUFNI=6
- CUSHION=100K
- RMODE31=NONE (all from defaults) STATS (from both default and JOB)

and from JOB level elector:

- MAXBUFSP=120K
- MAXBUFND=15
- MAXBUFNI=20
- MSGLEVEL=10

You can alter the generic names in this member to represent test jobs in your shop. To activate this member, change CBS3PROC member GVBDBFON to read the CBS3OPTN member GVBSTART by specifying M=GVBSTART on the PROC statement.

See the explanation of program GVBDBFON in the section GVBDBFON - Activate CA-Hyper-Buf (see page 77) for further details.

Temporarily Suspend Buffering

The following control stream temporarily suspends buffering by CA Hyper-Buf.

```
SYSTEM_DEFAULTS         *  begin  defaults

       DEFAULTMODE=EXCLUDE      *  all  excluded
```

This control stream may be supplied to GVBREFR to suspend buffering by CA Hyper-Buf, or specified in the ISPF refresh panel. This control stream sets DEFAULTMODE to EXCLUDE. Since no INCLUDE mode control groups are specified, all OPEN's are excluded.

You can resume CA Hyper-Buf buffering at a later time by running GVBREFR with a full set of constraints.

Glossary

Constraint

Constraints are control statements that are supplied to GVBDBFON. The constraint controls place limits on the buffering performed by CA Hyper-Buf. This allows each installation to establish boundaries for the buffering variables which are honored by the CA Hyper-Buf buffer space calculation routine.

Level Selector

A level selector is a control statement in the GVBDBFON control stream. Each level selector determines the scope of the constraints which follow the selector. The valid level selectors are SMSCLASS, CLUSTER, JOB, PROGRAM, and PROGRAM/DDNAME.

LSR (Local Shared Resources)

A buffering scheme used by VSAM that allows multiple files to 'share' buffers and control blocks. (See also 'NSR').

MODE control

A MODE control is a control statement in the GVBDBFON control stream. Each MODE control starts a new mode control group and determines if the new group is an INCLUDE group or an EXCLUDE group.

MODE control group

A MODE control group is a group of control statements in the GVBDBFON control stream. Each group is delimited by MODE control statements. Each MODE control starts a new mode control group and determines the type of processing specified by the controls in this group. This mode control group is terminated at the next MODE control statement, or the end of the control stream, whichever comes first.

NSR (Non Shared Resources)

A buffering scheme used by VSAM that allocates separate buffers and control blocks for the exclusive use of each file that is OPENed. See also LSR.

UBF (User Buffering)

An option available to assembler language programs only, which allows the application to maintain control of the buffering process. CA Hyper-Buf cannot alter the number of buffers that these applications create.

Index

J

L

M

N

P

Q

R

S

T

U

V